Developer Relations Group

Hybrid Application Cookbooks

Multiplayer Internet Gaming

Intel Corporation
Developer Relations Group
www.intel.com/drg

Acknowledgments

Intel would like to thank the following companies and contributors for their assistance and information in writing this paper:

Dale Addink, Interactive Creations Inc. (ICI)
John Carmack, Id Software
John Cash, Id Software
Yu-Shen Ng, Mpath Interactive
Jeff Rothschild, Mpath Interactive
Jason Rubinstein, TEN
Jeff Kesselman, TEN
Bill Lipa, TEN

intel®

Table of Contents

1. So You Want to Write a Connected Multiplayer Game…

1.1 A Little History Lesson

At the beginning of 1996, connected applications were an interesting concept, but not much of a reality. At the time, Internet browsers were one of the few Internet applications available. These browsers simply shoved information down the wire, while this is an effective way to ultimately get information to the end user, it didn't add much opportunity for a rich experience. With the added horsepower that a powerful client can provide once the information gets there, the end-user's experience can be enhanced.

Early online games only included role-playing or simulator games. These types of games worked well in this online environment because they did not require real-time player updates and were limited to a small number of players. With the coming of age of the PC and the supporting OS', real-time, fast-action games are quickly becoming a reality.

Today, a new series of full-featured, robust clients are available. These new clients reduce the role of the servers by locally processing data, thereby limiting the amount of transmitted data and effectively increasing the speed of the game.

With the introduction of Windows95, DirectPlay, and Winsock several options have been
made available in the operating system-specifically, APIs that create an abstraction layer from the protocol implementation details which previously did not exist.

1.2 Where We're at Today

Online gaming is now growing out of its technical infancy to become a reality. Certainly many creative game developers have had dreams about what their ideal multiplayer gaming environment would be. Maybe something like this…

1.2.1 Capabilities of the Ideal Multiplayer "Dream" Game:

Audio/Video Phone - Whether I'm flying in an aircraft or spacecraft, or traveling through tunnels with a rocket launcher, I would be more immersed in a game if I could verbally taunt my enemy. With communications devices, I could tell my team where to move in for an ambush, to retreat if we are losing, or just let them know my status. How much more fluid would the game be? Take for example about a year ago I was playing Descent II with some co-workers. Alliances were formed and each team would gang up on each other. I conference-called my team over analog phones one day to share my ideas on strategy. Let's just say our opponents would always say "How the hell did you guys gang up on us so quickly?!" The added feature of continuous communication gave us the ability to warn each other as well as tell each other where to go.

Voice fonts - Impersonating special characters would add fun and humor, while making it easier to distinguish among different players. How embarrassing it would be to hear Alvin the chipmunk chuckling and saying "Hasta La Vista Baby" while toasting you!

Incredible high resolution visual 3D - We all have seen 3D animation, and those of us who have been blessed with playing Quake or Hellbender with a hot 3D accelerator card know that it is much more visually appealing (not to mention just plain easier to see). If every player had 3D, these games would be even more popular.

Spectators - It's always amazing to be able to watch someone who is a "master" at Quake hide around a corner, dart up to someone from behind, get their attention and then run circles around them while nailing them to the wall (literally!). If there was a 'ghost mode' capability in a game, we could fly around and learn the skills of these masters .

Surrounding 3D audio - If player A, who's being circled, could hear an opponent's footsteps around the corner, he or she could at least be able to turn around faster and get a couple of shots in before being blown to bits. This capability would surely add more realism to the game. Or if a player was flying a spacecraft, he could get a sense of where his enemy is from the 3D audio, turn accordingly, and find the opponent.

O.K., O.K. WAKE UP! Perhaps with a 100Mbit hub, a 400MHz processor, a $2000.00 3-D card, and virtual reality, one could establish the scenario we dreamed up above, or maybe a drive down the street to the “Multiplayer World Arcade” (at $40 a visit) could do it. But these are only a sampling of the types of capabilities we'd like to be able to offer on the home platform.

Unfortunately all developers wake up the to same low bandwidth, high latency infrastructure we’ve got today, making the implementation of these features extremely difficult. Referring to Figure 1-1, although there have been developments in ISDN and cable modems, broadband connectivity in homes will not make a significant appearance over the next 2-3 years.

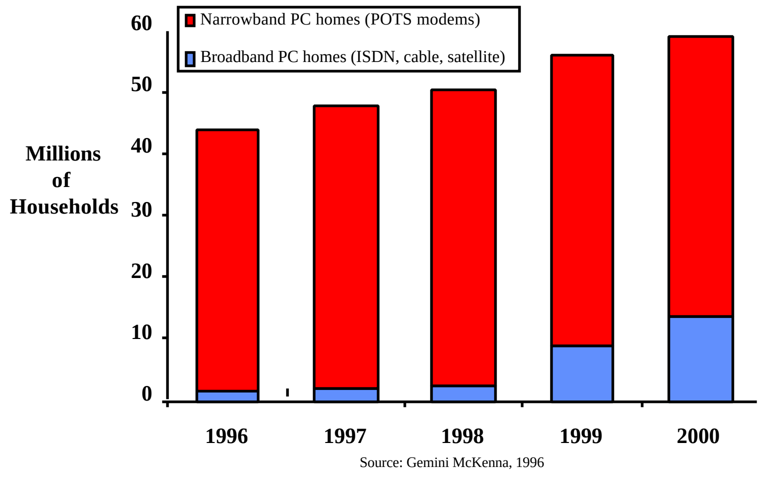


Figure 0-1: Narrow and Broadband modem saturation

Given what we have to work with, we may need to give up on some parts of these dreams for now. However, with the clever technologies that are currently available from various Internet technology vendors, we can add a little spice to multiplayer games, and start the trend of media-rich multiplayer gaming.

1.3 The Challenges for Tomorrow

This paper will address the issues involved in developing such games, and will give developers an introduction to the benefits and limitations of connected application architecture. It is not meant to be a relative checklist of features between various services and tool kits (which may change overnight from vendor to vendor). It ***is*** meant to give the application developer a framework of tradeoffs and issues to bear in mind. At the same time, this document highlights some of the key participants and vendors in this dynamic industry. We’ll also introduce what we consider to be the seven ‘Golden Rules’ of creating a multiplayer game, examine the many issues, technologies, tools of game development, and offer a few "tips and tricks."

2. HERE’S HOW YOU DO IT - DEVELOPMENT CONSIDERATIONS

Let’s not kid ourselves: It’s not easy to write a good Internet-enabled game given the problems associated with connecting and maintaining connections to the ‘net. However, the Internet does add significant value to the gaming environment: The chance to compete against another living person is far more compelling than playing an imaginary opponent.

Let’s take a look at some of the ‘Golden Rules’ involved in creating a good multi-user gaming environment.

2.1 Golden Rule 1: Design Your Game from the Ground Up

Developing multi-user Internet games involves two distinct aspects: content and connection. Each aspect provides a different level of sophistication for the game and both are equally important. A good network game without quality content will suffer just as badly as a game with good content and poor network support.

Designing the game to be multiplayer from the beginning may seem obvious to some, but others may consider reworking a single-player game model and its connection methodologies late in the development process to be a trivial matter. The ideal scenario is to be aware of what type of game you're trying to implement, know the risks and benefits of using that type of game, and then design it around the appropriate game model for best networking support.

There are three major game models for the multiplayer games that are used today: peer-to-peer, client/server, and fully connected. The game model and implementation details determine many of the design constraints involved in the development process. As a game is being developed, the designers must determine the available bandwidth, the protocols the game will support, and the essential features of the game. (See Figure 2.1.) The following sections will illustrate the differences between various game models and describe the technical issues of each.

GAME MODEL	PERFORMANCE	BANDWIDTH	LATENCY
Peer to Peer	Play at slowest PC	High	Low
Client/Server	Play at slowest connection	Medium	Medium
Fully Connected	Play at slowest connection	Low	Medium

Figure 0-1: Game Model Comparison

2.1.1 Peer-to-Peer Game Models

Peer-to-peer games are maintained by the game clients and not by a central game server. These games are also known as client state or session games.

In this model, each game client maintains a separate copy of the game itself. Actions taken in any one game must be communicated to all other clients in the game. See Figure 2-2. This ensures that each client's game state is identical. The game state is the status of the game at the current moment.

Because there is no server, each game client starts at the same time and with the same game state. One game client assumes the role of the initiator, or moderator, synchronizing communication between the other game clients. This is known as the clock-tick synchronization phase.

Typical peer-to-peer games are best played on Local Area Networks (LANs). LANs offer the proper communication medium for all connected game clients because of their wide bandwidth.

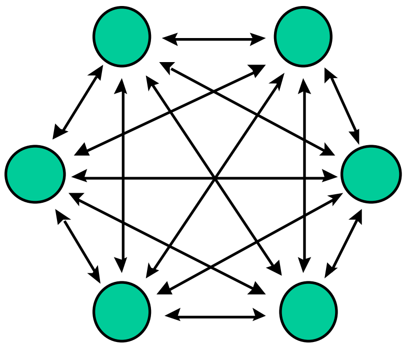


Figure 0-2: Peer-to-Peer Communications Model

2.1.1.1 Technical Peer-to-Peer Game Issues

Peer-to-peer games are restrictive in the number of users that are allowed to join. If a game client is allowed to join a game in progress, they must be notified of the status of the game. This requires that the joining game client request and receive an update from all the other clients currently in the game. The amount of data delivered to the joining client is proportional to the square of the number of connected game clients, and the amount of this data will increase dramatically as more clients enter the game. Looking at Figure 2-3, we can see that for ***n*** people:

$c = \frac{n(n-1)}{2}$, where ***c*** is the total number of connections

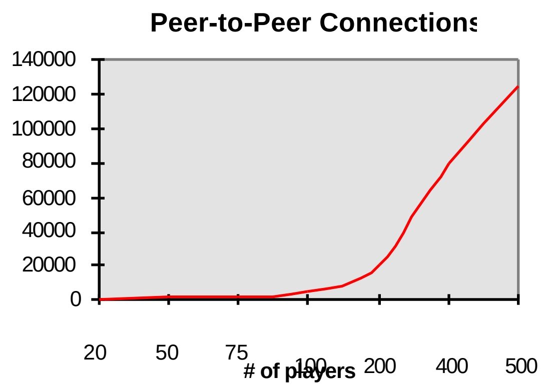


Figure 0-3: Peer-to-Peer Data Transfer Rates

Because of this problem, peer-to-peer games generally require that all game members be present at the clock-tick initialization phase (game startup). This prohibits the "drop-in" game client from joining a game in progress.

PC horsepower is another large factor in the performance of and participation in peer-to-peer games. In this model, game performance is restricted to a particular client's hardware. In other words, games will play at the speed of the slowest connected PC. For example, if Client A is played on a low-performance 60MHz Pentium Processor and Client B is played on a high-performance 200MHz Pentium Processor with MMX™ technology, the game will be played at the speed of Client A. This is because Client A cannot process the game information as quickly and efficiently as Client B. During the clock-tick initialization phase, Client A and B will negotiate the highest communication speed possible.

Yet another factor to consider with peer-to-peer games is that they are prone to user corruption, or cheating. Corruption can be defined as allowing a player to obtain information and change information about the game in progress. In the peer-to-peer model, a client will give unfiltered information to any other client that requests it. Therefore, it is a simple process to obtain information about another player, alter it, send it back during the next update, and thereby contaminate a particular game.

2.1.2 Client/Server Game Models

The model of choice for most multiuser Internet gaming (MIG) environments, is the client/server model. In the client/server game model, clients communicate with a dedicated game server instead of other individual game clients. The game server provides a two-way connection and facilitates all game state communication among the connected clients. The presence of a dedicated game server takes the majority of the message-passing and maintenance load off the clients, allowing large numbers of users to play and interact seamlessly with each other. Figure 2-4 depicts the communication model.

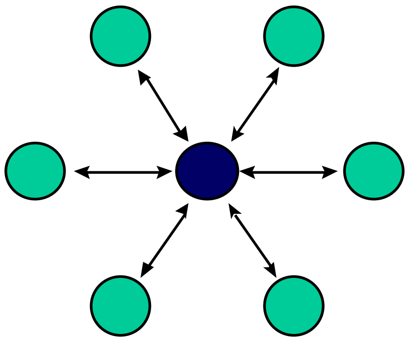


Figure 0-4: Client/Server Game Model

After the server has started a game, clients can solicit it for admission. The server will admit players until it reaches the player threshold limit. This limit is determined by bandwidth and CPU horsepower available.

There are two basic types of client/server game models: turn-based and real-time.

2.1.2.1 Turn-Based Games

Turn-based games allow players make plays in a sequential order. This model works very well for strategy games such as ***Tic-Tac-Toe*** or ***Monopoly***. In these games, each player waits and thinks about a move before they act upon it. As each player finishes his or her turn, the game server sends an update to all the other game clients. The exact arrival time of the updates is not critical to the playability of the game, which makes these games playable on networks restricted by high latencies and narrow bandwidth.

2.1.2.2 Real-Time Games

Real-time games allow players to move and act simultaneously. If a player makes any type of change to the game he or she must notify the game server. The playability of the game is dependent on efficient data transmission between the clients and server.

As the scale of the game increases, game servers are required to find alternatives to reduce the amount of data transmission that occurs during a game. One method, based on the assumption that Player 1 won't have to know about Player 2 if they're in different rooms, uses sorting algorithms to determine which information to send to which client. The sorting algorithms classify players according to their current position, take the coordinates from a player's position, and compare the player's coordinates to those of other players in the game.

Typical games servers will compute 16 or 32 of the closest (coordinate-wise) game players. This number is completely dependent upon the processing horsepower available to a game server and the speed of the sorting algorithm. If the total number of players in the game is less than this limit, information regarding all of the players may be communicated to your game client. If the number of players exceeds this limit then only the 16 or 32 closest players will be calculated and their position coordinates will be transmitted to your client.

For example, Figure 2-4 shows the visibility for Player 1 in a game. In this example there are seven players. Currently the game server is sending Player 1 positional information regarding Players 2, 3 and 4. The other three players (5, 6, and 7) are outside of the fixed visibility radius.

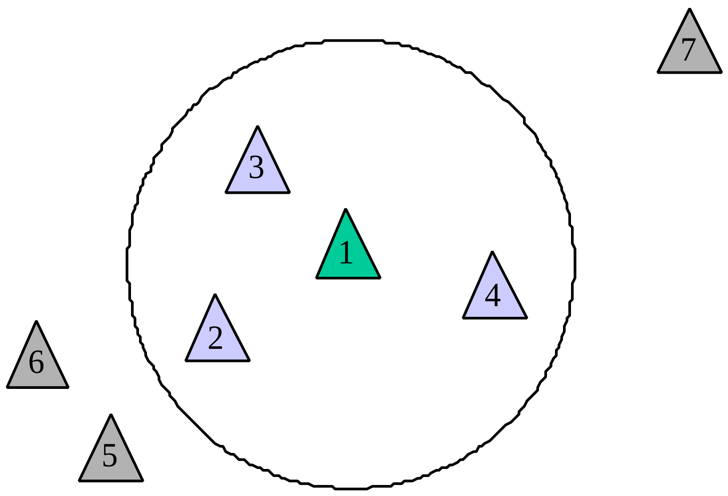


Figure 0-5: Game Visibility for One Game Player

As users join games already in progress, their game client queries the game server for game information. The game server will compute the ***n*** closest players, according to their visibility radius, and relay information regarding their position and assets.

2.1.2.3 Technical Client/Server Game Issues

Because all game client communication is facilitated by the game server, the amount of data that is exchanged in games increases linearly with the number of game clients. (See Figure 2-6.) For example, if Client A joins Game 1, Client A will only communicate with the game server for Game 1, not the individual game clients themselves. This efficient communication model allows server state games to potentially accommodate hundreds of simultaneous users.

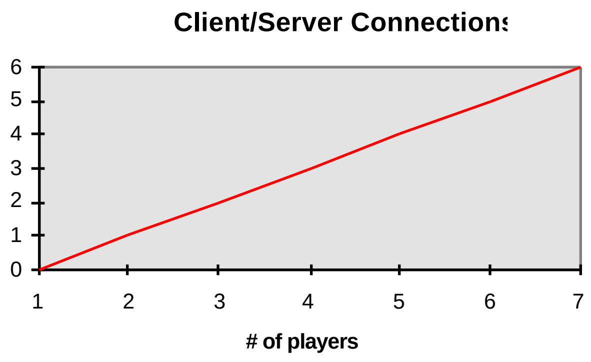


Figure 0-6: Client/Server Data Transfer Rates

Client/server games are not bound by the slowest PC involved in the game like the peer-to-peer games are, but instead perform at the rate of the slowest connection to the game server. A single high-latency connection will render a real-time game unplayable. There are several solutions available to work around this issue; they will be explored in the following sections.

Unlike client state games, server state games are not prone to user corruption. All of the player information is stored in a secure database on the game server, which makes it very difficult for a user to improperly adjust an individual player's capabilities.

2.1.3 Fully Connected Game Models

Fully connected game models incorporate features found in both peer-to-peer and client/server game models. For example, message-passing among clients may be handled in a peer-to-peer manner, while player statistics are directed to the game server as in the client/server model.

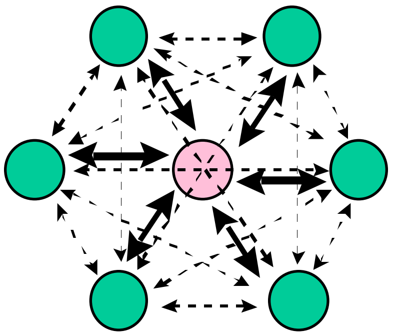


Figure 0-7: Fully Connected Communication Model

2.1.3.1 Technical Fully Connected Game Issues

In this game model, the clients take on more of the communication burden. This lessens the server's responsibility for handling all incoming and outgoing messages, and allows it to concentrate on the game state information. Because of this, and given a large amount of bandwidth, the server will be able to accommodate more potential simultaneous players.

The vital player statistics are still stored in a secure database on the server to prevent user corruption, as in the client/server model.

Individual direct messages (such as text or audio chats specifically to another player) can be sent directly and circumvent the server altogether.

However, the Fully connected model will require a special scheme to check for network congestion as multiple packets will be coming from multiple sources (other clients as well as the server). That being the case, the server will not have complete control over the message-passing, and won't be able to scale its messages appropriately based on network congestion.

2.1.4 Tips

TIPS FOR CHOOSING THE GAME MODEL

- **Real Time Games:**
  - **Decouple communications from the Game**
  - **Use fastest possible sorting algorithms**
- **Turn Based Games:**
  - **Restrict number of players**

2.2 Golden Rule 2: Know your Market - Choose the Right Connection

Once you know your game model, you need to decide how many users you intend to allow to connect to your game, as well as ***how*** you intend to allow them to connect. Both of these factors will influence the types of connection technology you use. Will you be running your own servers? Do you plan on using an Online Service Provider (OSP)? Will you allow anyone to run a server? Will they play regionally, nationally, internationally? No matter what your answer is to all these questions, there is one thing that must be made clear: No matter what type of communication model is used, it must be an easy process. Whether they go through the WWW with a point and click interface, or use a direct-dial scheme, end-users require that it be easy to get online, to find some people to play with, and to begin the game in less than 5 minutes.

In the following sections, we'll discuss the game types with respect to developing games of different scales and for different numbers of players. We'll discuss the need for an easy connection process and explain why it's a requirement. Finally, we'll give software developers a technical overview of what they need to know about the network plumbing. We've designed this section to provide a linear guide through the connection world. We'll start off with a low-level introduction to network protocols and move our way up the data packet food chain to APIs and Windows sockets. We'll also discuss high-level connection solutions offered by many OSPs, and conclude with some ideas on a simple connection model.

2.2.1 How Many Can Play?

Depending on what type of game is being designed, you must make a decision on how many players are allowed to connect to a session. This decision helps define both the scope and the complexity of the game. Figure 2-8 describes some typical game models versus the game scale.

Scale	# of Players	Comments
Small Scale	2 - 8	Peer-to-Peer
		Real-time
		Sensitive to bandwidth
		Scalability limited by available bandwidth
		Clients broadcast transmissions to each other
Large Scale	8 - 100	Client/Server or Fully Connected
		Real-time, Role Playing or Simulators
		Geographic subdivision or filtering of data to reduce modem overhead (sorting)
		Server CPU limits scalability
		Server multicasts transmissions to clients
Super Large Scale	100 - 1,000+	Client/Server or Fully Connected games
		Role Playing or Simulator games
		Requires multiple CPU servers
		Requires large bandwidth

Figure 0-8: Game Models

Game Type	Game Model	Scale	Popular Examples
Real-time	Client/Server or Fully Connected	Large Scale	***Quake***
Simulator	Client/Server	Large Scale	***Terminal Velocity***
Turn-based	Peer-to-Peer	Small Scale	***Monopoly***
Role Playing	Client/Server or Fully Connected	Super Large Scale	***Diablo***

Figure 0-9: Characteristics of Game Models

2.2.2 Keep the Connection Simple

Now, with all these players trying to get connected and play, what's the biggest hurdle to overcome? It's the connection process in general. If users can't get connected in less than 5 minutes they begin to get frustrated. After all, the point is to spend the time playing the game, not trying to connect. There are really two groups to consider here: players who want to connect to someone they know (usually in a peer-to-peer environment), and players who simply want to get online and play. In either case, the connection model must be as simple as possible.

For the people who want to play point-to-point, it becomes very important to hide the dialing mechanism. Bury it in a clean user interface (possibly the WWW, something most users are familiar with). Consider this typical scenario. User A and User B first call each other over the phone and decide they want to play a game. Then they dial their respective ISPs, and each gets a unique IP address for that session. The players then must exchange that information while they're online (and if they hang up in order to call each other and exchange their IPs, they might get new ones the next time they log on), so they have to email the information, then go through the game's connection process, and finally start playing. This is a discouraging process indeed.

When someone just wants to get online and begin playing with anyone, they have a couple of options. Many games allow users to set up their own servers, but the problem with this is that it's difficult to publicize their availability to the rest of the gaming community. In addition, unless you have a dedicated connection to the 'net, the IP address of the game server will be changed every time you start the server (going through an ISP which dynamically assigns IP addresses). Another solution is to make use of the dedicated connection solutions and enhanced feature sets available from numerous Online Service Providers (OSPs).

2.2.3 Low-Level Connectivity Protocols

This section is included to eliminate any confusion regarding the network protocols that are used in multiplayer games. Today, the Internet only supports Transmission Control Protocol/Internet Protocol (TCP/IP) and User Datagram Protocol/Internet Protocol (UDP/IP). Of the protocols that will accept packet loss and still retain the maximum amount of functionality, TCP/IP is the only one that supports large scale networks. The Internet Packet Exchange/Sequential Packet Exchange (IPX/SPX) protocol is only supported on LAN connections. Many OSPs will not provide future support of the IPX/SPX protocol for their users, so we'll not discuss IPX here.

2.2.3.1 Internet Protocol

Let's start with the basics. The Internet Protocol (IP) can be represented generally as a four-stage model in which each stage is adapted to a standard provided by the International Standards Organization (ISO). (See Figure 2-10.) These building blocks constitute what is called the ***protocol stack.***

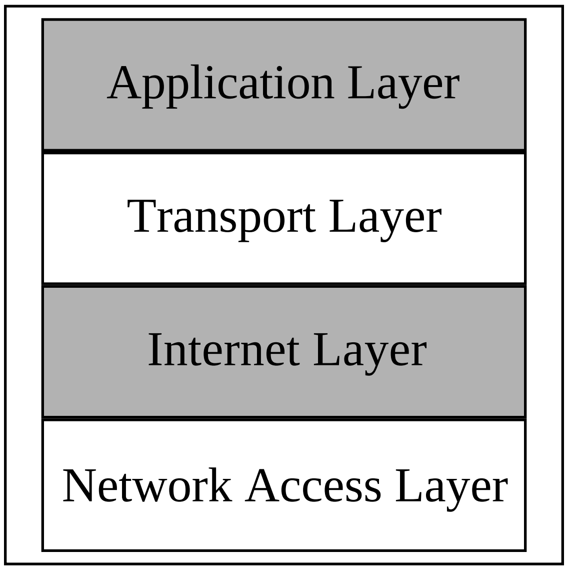


Figure 0-10: Layers in the TCP/IP Protocol Architecture Model

The IP is the core connection protocol that Internet clients and Internet servers user to exchange data. For example, as a Web browser downloads an HTML file from a Web server, the IP stack on the server will divide the file into one or more datagrams and transmit the data. The IP stack on the client side is responsible for reassembling the individual datagrams to create the original HTML file.

As a software developer you will only be interested in the Application and the Transport Layers. The Application layer includes the interface that allows your application to communicate with other applications.

The Transport layer hosts two of the most important datagram protocols - Transmission Control Protocol (TCP) and User Datagram Protocol (UDP) (See Figure 2-11.) TCP and UDP represent connection-oriented (you establish a connection, transmit the data, and then terminate the connection) and connectionless (each connection carries the full destination address and is routed independently of each other) services, respectively. These two protocols are chartered with managing the packaging of data into datagrams that get routed on different paths over the Internet and reassembled at their destination. The Internet Protocol handles the address part of each data packet so that it is routed to the right destination.

Issue	TCP	UDP
Initial Setup	Required	Not Possible
Destination Address	Only needed during setup	Needed on Every Packet
Packet Sequencing	Guaranteed	Not Guaranteed
Error Control	Provided by Network Layer	Provided by Transport Layer
Flow Control	Provided by Network Layer	Not Provided by Network Layer
Data Packet Overhead	24 Bytes	8 Bytes

Figure 0-11: Major Differences Between TCP and UDP

2.2.3.1.1 TCP

Applications that use the Transmission Control Protocol require that their data be delivered in the order in which it was sent. This requires additional overhead be incorporated into the data transmission scenario. Going back to the web browser example, if the web client receives datagram 1 and then receives datagram 3, it stops processing the data packets and requests that the server resend datagram 2. This process takes a fair amount of time, but it does ensure that all of the datagrams are received and reassembled in proper order. TCP should only be used to send streams of data that must be received in sequential order such as voice or video data.

2.2.3.1.2 UDP

The User Datagram Protocol (UDP) was designed to facilitate message-passing between clients. The major difference between TCP and UDP is that UDP is unreliable. UDP allows messages to be exchanged over the network with minimal protocol overhead. These messages come in two categories: essential and non-essential.

Essential messages can be used to inform all game players when a player dies or when the game ends. To ensure that these messages are received, the software developer must develop a reliable message handler. The non-essential messages do not need to be addressed; the game will continue regardless of whether or not they reach their intended destination. If the messages are critical to the continuity of the game, you must use your own reliable message delivery mechanism.

2.2.3.2 RSVP

Another protocol which should be mentioned is the Resource Reservation Protocol (RSVP), which was designed to allow routers to reserve bandwidth for applications that request it by decoding IP traffic and then encoding it with a priority value. While RSVP is not widely used at this point (it requires router software updates) it is an important technology to be aware of.

2.2.3.2.1 PC-RSVP from Intel

Intel's PC-RSVP is a software layer that provides reserved bandwidth to network applications. Applications call PC-RSVP using the Quality of Service extensions in Winsock2 to request bandwidth from the network. The network grants the request and from then on, the application is able to get the bandwidth it asked for, no matter how congested it gets.

With this reservation, an application can get enough bandwidth on most networks to run data-intensive content, such as video, sound or animation streams between two computers. The result is a smooth experience: The content flows free of any delay or jerkiness because no other network traffic can disturb it.

2.2.3.2.2 Technical RSVP Issues

For a complete discussion on RSVP, see: ***http://www.cisco.com/warp/public/724/4.html***
http://www.intel.com/ial/rsvp

2.2.4 Low Level Connection Solutions

Now that we have built a foundation of what is going on behind the scenes, we are ready to see some of the development tools available to end users, specifically, the low level connection solutions.

2.2.4.1 Winsock

The network programming interface for Microsoft's family of Windows products is known as Winsock. This interface takes advantage of the message-handling features of Windows. (See Figure 2-12.) This programming interface allows developers to create both client and server state games. The Winsock specification provides a single Application Programmer Interface (API) that standardizes the network software development process and abstracts the connection issues for the software developer.

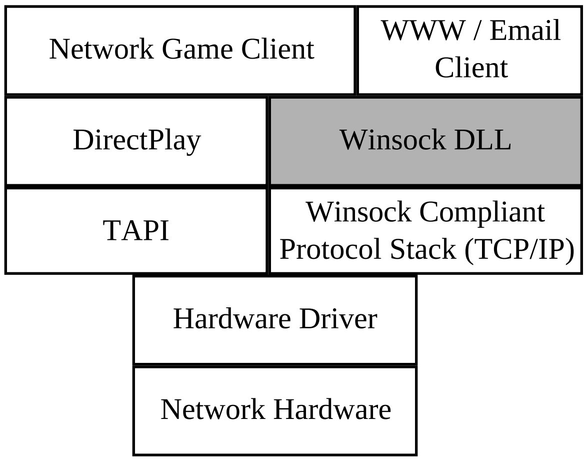


Figure 0-12: Winsock Network Architecture

2.2.4.2 Winsock 2

While preserving strict backwards compatibility, Windows Sockets 2 extends today's immensely popular WinSock version 1.1 interface for TCP/IP networks to provide a protocol-independent interface that is optimized to support real-time multimedia communications and is able to work across all kinds of communications infrastructures.

Windows Sockets 2 is a network programming interface at the transport level in the OSI reference model. Designed to the WOSA (Windows Open Services Architecture) format, it provides both an API for applications developers and a Service Provider Interface (SPI) for transport stack vendors.

2.2.4.2.1 Features and Benefits

Windows Sockets 2 provides numerous enhancements to the original specification:

Multiprotocol Support
Windows Sockets 2 significantly extends the interface by providing applications with transport independence across a wide range of networks and communications media types such as TCP/IP, OSI or IPX/SPX. Applications can now be developed for portability across a variety of transports with a common interface for all types of network stacks and communication media.

Quality of Service
Windows Sockets 2 includes mechanisms for applications to negotiate quality of service with a network, thus facilitating its use for multimedia applications. Applications can discover and utilize the quality of service (bandwidth, latency, etc.) offered by underlying networks such as ATM, ISDN, RSVP (as a matter of fact, the same API is used for the RSVP QOS calls), and Cable. Virtually all of the major ATM vendors have announced plans for their desktop and server products to leverage WinSock 2.

Multipoint / Multicast
Windows Sockets 2 also provides generic support for both multipoint and multicast which allows applications to discover and use capabilities like IP multicast and ATM point-to-multipoint in a protocol-independent manner. Multipoint and multicast will enable the fully connected gaming model to become a reality, taking much of the dependency off of the servers.

Real-Time Multimedia Communications
Windows Sockets 2 was specifically engineered to enable a new breed of applications that employ Real-time Multimedia Communications (RMC), such as video conferencing. Besides Quality of Service, there are several other features in WinSock 2 that assist applications with RMC such as overlapped I/O, circular queues, and buffer flushing.

Layered Provider Architecture
Windows Sockets 2 includes provisions for inserting layers between the WinSock 2 DLL and the underlying protocol stack that communicates over the wire. A layered protocol could be used to implement secure authentication and encryption for billing or security issues, and many third parties are creating plugins for this purpose.

Event Object Based Notification
In addition to the window message based asynchronous notification of network events in version 1.1, Windows Sockets 2 offers event object based notification in order to facilitate development of application servers and deamons which require no Windows GUI components.

For a complete discussion on the Winsock2 implementation, see: ***http://www.stardust.com***

2.2.4.3 Microsoft DirectX

The Microsoft® DirectX™ SDK includes a set of APIs that provide the resources to design real-time applications. DirectX was developed to enhance the performance of applications running under the Microsoft Windows® operating system so that they rival or exceed the performance of applications running in the MS-DOS® operating system or on game consoles. This SDK was released to promote game development for Windows by providing a standardized development environment.

Microsoft's DirectPlay® is a particular API included in the DirectX SDK that allows your application to access communication services built into Windows. (See Figure 2-14.) DirectPlay provides the mechanism necessary for efficient communication among applications. The DirectPlay APIs are intended to be independent of the underlying transport, protocol, or online service.

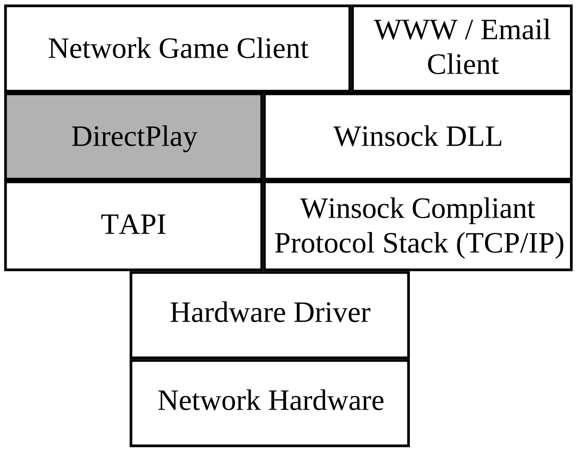


Figure 0-13: DirectPlay Network Architecture

DirectPlay uses a peer-to-peer paradigm for communication among clients. One game client will designate itself as the game host. All other game clients will register themselves with the host and receive a unique identification number. A message sent from one client to another is routed to the host and then to the intended client. The message-handling APIs have been designed to minimize the overhead required for the host to receive, interpret, and route client messages.

The DirectPlay Component Object Model (COM) contains two API functions that are used to enumerate DirectPlay servers and create DirectPlay objects: **DirectPlayEnumerate()** and **DirectPlayCreate()**. Typically, an application will first enumerate all available DirectPlay servers. The user will then select a particular server from the list of available network service providers and create a DirectPlay object based on that server.

For a complete discussion on the DirectPlay architecture and implementation, see:
http://www.microsoft.com/developer/tech/dx3doc/

2.2.5 High Level Connection Solutions

Software developers who wish to add multiplayer capabilities to a game have other options than that of using the low-level connection solutions. Many online service providers (OSP) cater to developers who are seeking assistance in this area. These OSPs will provide APIs, libraries, and, in some cases, porting assistance to port single player games to the multiuser environment. This method of integration ensures that the game will perform optimally for a particular network architecture.

2.2.5.1 OSP Services

Online Service Providers attempt to provide the best online gaming experience by offering both developer and user services to the PC gaming community. OSPs focus their attention on the game designs rather than on the game content.

In addition to providing the hottest multiplayer games, each OSP has a different strategy for providing low-latency services. These strategies include providing proprietary APIs and, in some cases, assisting developers in adapting multiplayer functionality to current single-player PC games. This, in addition to the long list of services they provide to the user, make them a very attractive option for the many game developers looking to enter the mutiuser world. A few examples of quality user services include:

Local phone numbers for easy dial-up access
Guaranteed low latencies
Scalable and reliable network service
Storage space for game databases
Secure network protocols for billing
Latency checks for connections

Full-duplex modems for voice-capable chat rooms
Game arenas that accommodate hundreds of players
Persistent databases for continuous game play
Tournament servers fed by game databases
Fault-tolerance allowing games to gracefully handle intentional and unintentional game exits
Automated game-patch update program
Online beta programs to help software developers test new games

OSPs will generally require users to obtain the following components before access to the game services can be granted.

Internet connection with at least a 14.4 Kbps modem
User Interface application
Game-independent enabler
Either the shareware or the retail version of the game they wish to play (the game must also be offered by the OSP)

These online gaming services offers new depth to the game-playing experience. These services are a newcomer to the industry, but they have a very marketable concept. As the number of online users continues to grow, the demand for quality services that offer the hottest games will increase at an aggressive pace. For more information on three such OSPs, more information can be found with:

Mpath Interactive	***http://www.mpath.com***
Total Entertainment Network (TEN)	***http://www.ten.net***
America Online	***http://www.aol.com***

2.2.6 Tips

TIPS FOR CHOOSING THE RIGHT CONNECTION

Keep the Connection Simple!
Get a More Direct Connection

2.3 Golden Rule 3: Anything That Can Go Wrong With Communications, Will

What if you were playing an 8-player game when suddenly you were kicked out because someone lost their connection? Pretty annoying eh? When designing a multiplayer game you have to consider and plan for all contingencies. A few examples:

The game must gracefully continue if someone drops out unexpectedly
The game must detect cheaters whenever possible (players intercepting and changing packets, people unplugging their modem lines so they won't get killed)
The game must check for latency problems and scale itself appropriately so that game play can continue at a reasonable level

These are only a few examples of the types of things we're talking about. In general, the title of this section is very appropriate, and the trick is to see what those "anythings" are and have plans in place to deal with them. Much of this preparation comes out of having a rigorous testing plan.

2.3.1 Fault Tolerance

Computers will crash. This fact is inevitable. Game servers must be able to gracefully recover after a game client has been intentionally or unintentionally removed. An intentional removal occurs when the player decides they do not like the course of the game (i.e. they're going to lose) and they pull their modem cable. Unintentional removal occurs if the game client crashes and stops responding to the server. In both cases the server will cease to receive game updates from the client. The server must be designed to detect these changes and gracefully remove the players.

2.3.1.1 Up Time

Server up time corresponds to the stability of the server. Take for example a persistent game server that is up 99.5% of the time. If a crash represents a downtime of 1 minute, then there will be a crash every 3.3 hours. This means for every 3.3 hours a game is up, the game is down for 1 minute.

$$\frac{1\min crash}{0.005 crash} = 200\min$$

$$200\min = 33hrs.$$

2.3.2 Tips

TIPS FOR PLANNING FOR COMMUNICATION PROBLEMS

- Keep the Connection Simple to Begin With
- Get a More Direct Connection
- Take as Much out of the Equation as Possible
- Plan for Anything!

2.4 Golden Rule 4: Balance The Bandwidth Budget

The golden rule of bandwidth states that the greater the number of people participating in something, the slower that something gets. Have you've ever tried to suck a milkshake through a very thin straw? Well, this is the same principle. Developers need to be aware of the content they're trying to send through the straw. Games that plan to accommodate thousands of simultaneous users, or games that depend on large amounts of content being downloaded or streamed onto the PC are difficult, if not impossible, to implement with these bandwidth limitations.

So, you ask, what's the real problem? I've got a 28.8 Kbps modem attached to my ISP, why should any multiplayer game I play suffer or be any different than the stand-alone version? It's very simple--it's the connection. Every multiplayer game has a limited amount of bandwidth available. Consider that the total bandwidth is represented by a cake. Every player that wants to join the game eats a slice of the cake. There are several factors that will determine the size of the player's slice of cake. The largest contributor is modem speed. A game that would normally feed eight 28.8 Kbps modems might only be able to feed four 14.4 Kbps modems.

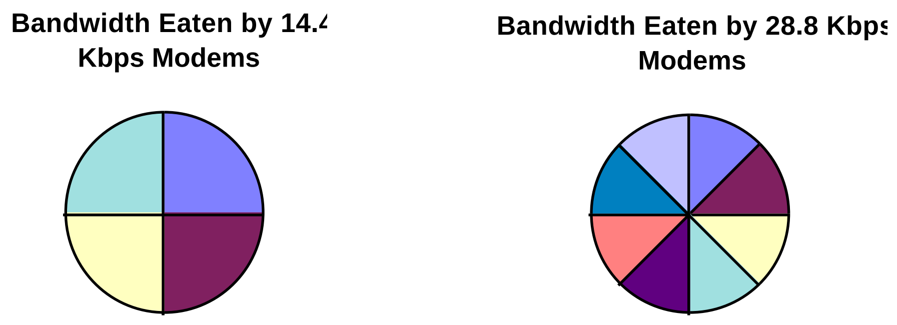


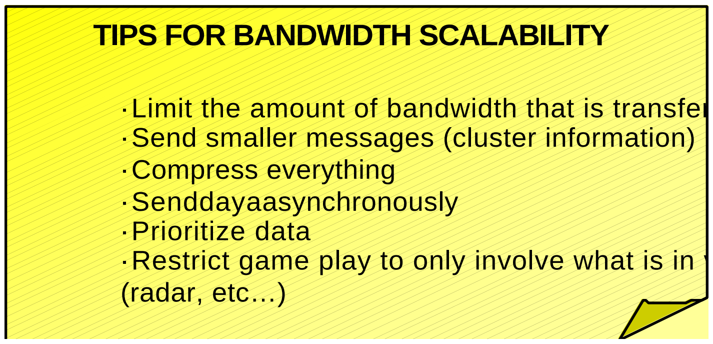
Figure 0-14: The Bandwidth Budget

The slower your modem, the more bandwidth you will require (because more transmissions will have to be made to get the same amount of data across the connection). Whatever you do, make sure you're only sending the necessary information, because that straw isn't getting any bigger anytime soon. The amount of available bandwidth is increasing very slowly.

2.4.1 Bandwidth Scalability

When you consider the wide variety of connection choices users have today, you see that allowing the user to scale the bandwidth based on the particular connection is a requirement. Allowing clients and/or servers to change the rate at which updates are sent out can make or break a high-action game. Also, correctly prioritizing your data (game logic, 3-D, audio, video), and putting the data in a need-based hierarchical order as the bandwidth reaches its limit will help in the overall game play. You may be able to lose some of the media, but retain the responsiveness and keep the player involved. Players in these environments consider the interaction and game play to be the most important thing, and the media secondary.

2.4.2 Tips



TIPS FOR BANDWIDTH SCALABILITY

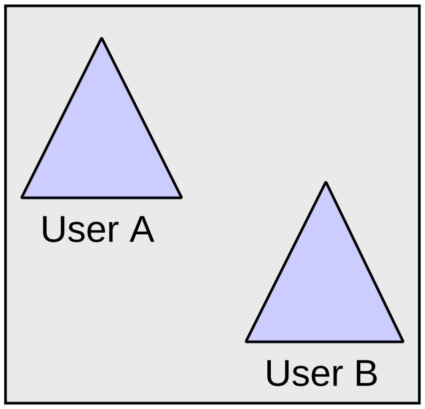
- Limit the amount of bandwidth that is transfe
- Send smaller messages (cluster information)
- Compress everything
- Senddayaasynchronously
- Prioritize data
- Restrict game play to only involve what is in v (radar, etc…)

2.5 Golden Rule 5: Beating the Clock - Latency Expectations

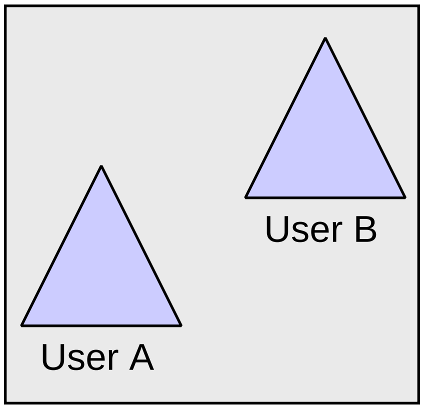
Let's face it: The term 'Internet Performance' is kind of like 'Jumbo Shrimp'. For most of us, it's a complete oxymoron. Connected multiplayer games are being developed because they are an obvious solution to the problem of getting connected to the Internet and allowing for multiplayer capabilities, while still allowing rich multimedia content from the PC side. The Internet that allows us to get online and play with our neighbors is the same Internet that also limits the features of whatever game we're playing.

A little history here: The Internet is composed of networks which are composed of more networks which are composed of even more networks. This is no news to most of us. The problem lies in the fact that the connection technologies people use to connect to this network are not keeping up with the technology itself. Modem technology is getting faster and faster, but there is a limit to the bandwidth that standard telephone services can provide. Today's 28.8Kbps and 33.6Kbps modems are rapidly approaching that limit.

Latency is the round-trip time it takes for a single data packet of information to be sent from the client to the server and back to the client. To the end user, it basically reveals itself in how the game is viewed. For example, in a flight simulator, if you have two players flying side by side in reality, and one of them has an abnormal latency spike, the two pilots may see the following:



What User A Sees



What User B Sees

Figure 0-15: A Latency Example

There are literally hundreds of paths through which a data packet could be routed from point A to point B over the Internet, and since we can't know which path a data packet will take, we can't know the latency for that data packet. In order to better understand what we're up against here it is important to understand two points.

The first point is that just because two computers on the Internet are located in close geographic proximity, that does not mean that they are located in close network proximity. Your data packets can hop many networks in the course of their travels.

A useful utility to count the number of network hops between your PC and your Internet destination is the ***TRACEROUTE*** utility found in most TCP/IP network stacks. This utility will report the number of network hops, and the latencies for each hop, that your packets encounter before reaching their destination. The reported latencies are not important here. The hops indicate the path that you will take to get to your intended destination.

For the purposes of this example, we used the ***TRACERT.EXE*** utility included in Microsoft's Windows95 TCP/IP protocol stack. Figure 2-17 reports the network hops encountered as we traced the route from a location in Hillsboro, Oregon to a web server in downtown Portland, Oregon. The physical distance from these two points is approximately 20 miles. The data packet that we traced however, traveled several thousands of miles! The point to remember here is that more hops equals a higher end-to-end latency.

Tracing route to www.cs.pdx.edu [204.203.64.13] over a maximum of 30 hops:

Hops	Latency (ms)	Latency (ms)	Latency (ms)	Destination - Addresses Resolved
1	10	10	15	134.134.246.251
2	10	10	10	uunet-f0.jf.intel.com [192.198.129.134]
3	31	203	16	901.Hssi3-0.GW1.POR1.ALTER.NET [137.39.164.1]
4	10	16	15	134.Hssi3-0.CR2.SEA1.Alter.Net [137.39.58.62]
5	32	31	31	110.Hssi6-0.CR21.SCL1.Alter.Net [137.39.58.50]
6	*	31	31	312.atm6-0.sr1.sc11.alter.net [130.39.13.157]
7	*	47	31	core5-hssi5-0.SanFrancisco.mci.net [206.157.77.73]
8	47	47	47	border1-fddi-0.SanFrancisco.mci.net [206.70.2.162]
9	47	47	46	border1-fddi-0.SanFrancisco.mci.net [206.70.2.162]
10	*	47	62	ocate.SanFrancisco.mci.net [204.70.32.6]
11	47	*	*	eugene-hub.nero.net [207.98.64.5]
12	62	78	63	beaverton-hub.nero.net [207.98.64.18]
13	62	*	62	psu-gw.nero.net [207.98.64.18]

14	78	62	*	207.98.126.20
15	*	47	94	sirius.cs.pdx.edu [204.203.64.13]

Figure 0-16: Network hops encountered from Hillsboro, Oregon to Portland, Oregon

The second point pertains to the basics of modem technology. Today, analog modem connections are the most popular method for residential users to connect to the Internet. The drawback for this type of connection is the latency the modems introduce.

An analog modem modulates digital signals from a computer or other digital device to analog signals for a conventional copper twisted-pair telephone line and demodulates the analog signal and converts it to a digital signal for the digital device. See Figure 2-18.

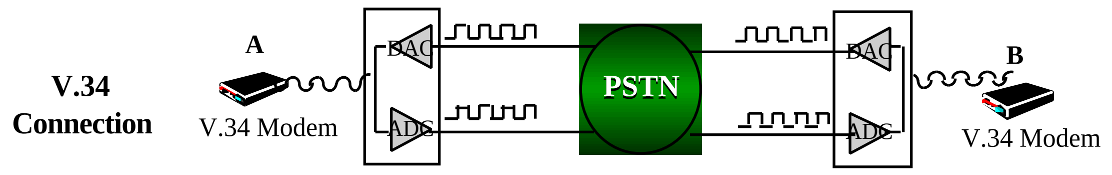


Figure 0-17: V.34 Modem Connection Model

To better understand the latencies encountered over a typical connection, you can use the common ***PING.EXE*** utility found with most TCP/IP protocol stacks. This will show you an example of the latencies caused by modem hardware.

A user issues a ***PING*** command to determine the latency for a particular ISP. The ping command uses the TCP/IP protocol that requires an acknowledgment from the destination server. This example assumes that the ISP server was using an analog modem. This is not always the case. Most ISPs have digital transmission systems (e.g., T1, T3, etc.) that do not introduce significant delays.

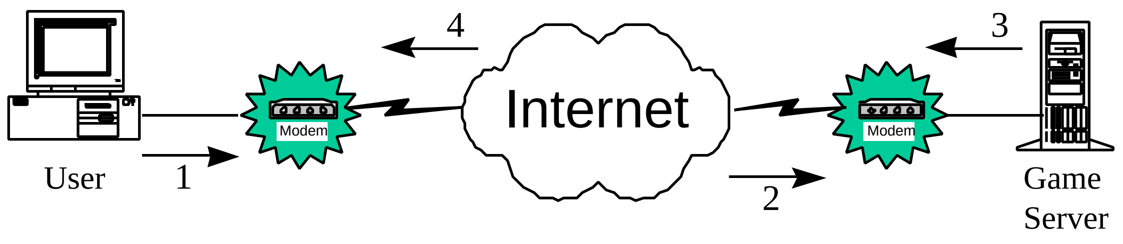


Figure 0-18: Typical Internet Communication

The first stage is represented by the user's modem. In this stage the ***PING*** command is issued by the user's PC and sent through the user's modem. The modem converts the digital information from the PC into analog data and transmits it to the ISP's Internet address.

The second phase is defined as the ISP's modem receives an analog data representation of the ***PING*** command and demodulates it to digital information. The request for acknowledgment is passed on to the ISP's server.

After the ISP has processed the incoming PING request from the user, it is ready to respond. In stage three, the ISP's server sends the acknowledgment to it's modem. The modem modulates the digital signal to analog data and transmits the data back to the source Internet address.
The last stage is represented by the User's modem receiving the PING reply, in the form of analog data, from the ISP's modem. The data is modulated back to digital information and relayed to the User's PC.

Each modem that the data passed through (Figure 2-18) required a specific amount of time to convert incoming or outgoing data. The average latency for a 28.8 Kbps modem is 20-30 milliseconds. On the individual basis this latency is small, but as we have demonstrated there are four modems involved for each transmission. The accumulation of these latencies introduce a total of 80-120 milliseconds of delay. Considering that a fast-action game will not tolerate latencies greater than 250-millisecond, modems consume a large portion of the bandwidth budget. If we combine that with some other contributors to the latency problem, we see that if we require a low latency connection for a fast-action game (say, 250ms), the playability of the game is in jeopardy.

Contribution	v.32bis (14.4 Kbps)	v.34 (28.8 kbps)
Modem Overhead (4 Modems)	~70 ms	80 - 120 ms
Data on Wire (20 Byte Packet)	~36 ms	~18 ms
WAN (National Round Trip)	~100 ms	~100 ms
Server / OS	~15 ms	~15 ms
Total	~221 ms	~218 ms

Figure 0-19: Latency Contributions in the US

The main point is that when the game is in it's early phases of development and the game model is being chosen, you must be aware of the types of latencies you're going to expect and tolerate. For example, any game you choose might typically see a 250ms latency, a fast-action client/server twitch game played regionally may not tolerate more than a 1.5 second latency before the game becomes unplayable. A fully connected simulator game may be able to accept a much higher latency (possibly up to 5 seconds), before it will become unplayable. You have to consider who will be playing the game, what their connection scheme will be, how often you'll check for latency problems, and how you intend to degrade the game gracefully. These are all considerations that the developer must take into account.

2.5.1 Tips

TIPS FOR LATENCY SCALEABILITY

· Turn off all modem compression / error checking
· Limit the amount of bandwidth that is transferred
· Avoid routers
· Plan for your maximum latency
· Gracefully degrade gameplay while waiting
· Abort gracefully if an outage persists
· Don't use the all-or-nothing events (1-shot kills)
· Make use of predictability
· Send daya asynchronously
· Cluster repetitious game events
· Restrict game play to involve only that which is in view (radar, etc…)

2.6 Golden Rule 6: Scale to the Platform

No two users out there will have the same configuration and connection scheme. User A will have a 60MHz Pentium® Processor with a VGA video card, connecting via a 14.4Kbps modem. User B may have a 200MHz Pentium® Processor with MMX technology, a high-end 3-D accelerator, and connecting via a 56Kbps modem. If these two connect and begin playing, what should the end result be? Should the game be scaled back so as that it runs well at the slowest connection speed? Should it be scaled for the slowest processor? These are questions that must be asked when development begins.

Let's take a look at what the base computer systems today may look like:

Processor	**90 MHz Intel Pentium® Processor**
Monitor	**15" SVGA**
Modem	**28.8 Kbps**

Joystick	Yes
OS	Windows 95
Memory	16 Meg
CD	4X
Sound	Sound Card / Speakers
Video Card	2Meg - SVGA
Browser	Netscape or Internet Explorer

Figure 0-20: Typical Baseline Computer System

The high-end platforms may have the following:

Processor	200 MHz Intel Pentium® Processor with MMX technology
Monitor	17" SVGA
Modem	33.6 Kbps
Joystick	Digital
OS	Windows 95
Memory	32 Meg
CD	12X
Sound	Sound Card / Speakers
Video Card	4Meg - 3D-Accelerator
Browser	Netscape or Internet Explorer

Figure 0-21: Typical High End Computer System

These are the gaming platforms that are becoming predominant in the home. Knowing that, and given the rapid inclusion of Internet technology into consumer software applications, it's clear that building Internet support into applications is a prerequisite for a successful product. Companies have been doing this for some time now (well, at least for over a year), and the early examples offer valuable lessons in the do's and don'ts of building a fully connected game, not the least of which is the importance of including rich media (whether it be locally or remotely) and delivering it in a timely fashion.

2.6.1 Media Choices

With connection models and gaming models in mind, the developer can consider the possibility of various media in a multiplayer game. For example, while a continuous peer-to-peer model for video transmission with 10+ players is certainly not practical, perhaps an occasional "priority video transmission" to pass secret information would be possible and could add richness to the game. And though audio takes up less bandwidth, it is clear that a continuous connection would still hamper the game.

2.6.1.1 Video

Digital video has been a integral part of the multimedia revolution. But how can this rich piece of multimedia fit into the multi-gaming world? In fact the better question to ask is ***can*** it fit (literally) in the gaming world? Is the limitation purely Internet/network bandwidth related, or does part of the problem come from the performance of the processor? Certainly, if video transmission is to be possible, the game must not take up 100% of the processor.

An important variable to consider in order to figure out what is currently possible is the preferred video window size. Common video display sizes for streaming video over the Internet is QCIF size (Quarter Common Intermediate Format, 1/4 CIF, i.e., luminance information is coded at 144 lines and 176 pixels per line). This format makes sense for gaming since a larger "video window" means a smaller "game window". In addition, a QCIF size video would not require all of the processor's power. Thus decoding the video is not as much of a concern as how much video can be delivered through 28.8-33.6kbps modems. There are several clever attempts at streaming video from various vendors. Before mentioning them, let us first look at some fundamentals:

Take, for example, the delivery of a 10fps uncompressed QCIF size video. Uncompressed video gives the highest image quality. However, the higher the image quality, the higher the datarate is required. Figure 2.23 illustrates this example:

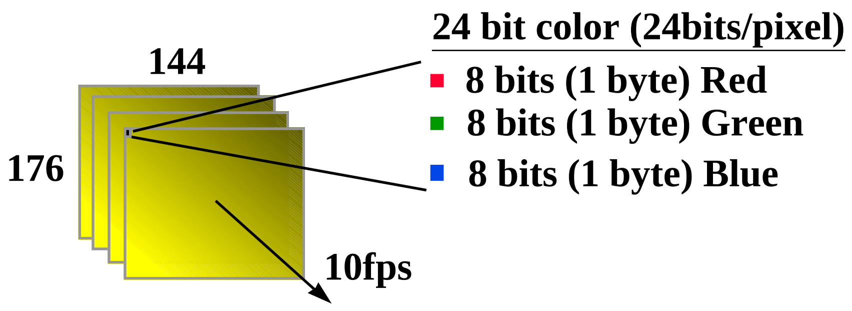


Figure 0-22

Without compression, an approximate datarate of this QCIF size video is:

$$\left(\frac{pixels}{line} x \frac{lines}{frame}\right) * \frac{frames}{\sec} * \frac{bits}{pixel} = \frac{bits}{\sec} \qquad \textbf{\textit{(2)}}$$

(144x176) x 10fps x 24bpp = **5940kbps**

It is very clear that a 5.9Mbit datarate video is very high and is not suitable for transmission over modems. Some codecs (**co**mpression-**dec**ompression algorithm**s**) presently offer a **~200:1** compression ratio which will crunch the video to an average datarate less than 10-30kbps. The tradeoff is a loss of video quality. In our example, the average datarate after compression is about 30kbps. It is important also to leave enough bandwidth for other data during a multiplayer session. Perhaps decreasing the framerate to 5fps (assuming a user is more interested in the game than a video window of his opponent, a high framerate transmission of the opponent is not required) would result in an average datarate of video to be 14.9kbps.
Streaming video technologies available in the market are:

ClearFusion - Iterated Systems	***www.iterated.com***
NetShow - Microsoft	***www.microsoft.com/netshow***
RealVideo - Progressive Networks	***www.real.com***
StreamWorks - Xing Technologies	***www.xingtech.com***
VDOLive - VDONet Corp.	***www.vdo.com***
VivoActive - Vivo Software Inc.	***www.vivo.com***
VXtreme - VXtreme Inc.	***www.vxtreme.com***

2.6.1.1.1 Video residing on CD

If a game does not need a live video feed of the opponent, another possibility is to have “taunt video clips” that are pre-encoded on the CDROM. Instead of seeing the real opponent, you see the character your opponent chose to play. The only time a video can be seen is when another player sends a taunt triggering the video to play. This doesn't requires video to be delivered over the Internet; the application just needs a small command to launch the appropriate video clip.

2.6.1.2 Audio

If there is video, there must be audio with it or else the video won't mean much (unless if you completely understand your opponents facial expressions). But suppose adding a video feature wasn’t relevant to the game? Leaving the video portion out leaves extra room to do more with audio. In addition, audio has the advantage in that, for the bandwidth available today, the audio streaming technologies can actually be put to practical use.

Let us do the bandwidth math again to show how practical it really is:

Parameter Change	Sample Rate	Freq. Resp.	Sample Size	SNR	Bit Rate	File Size (1min.)
Original	**44.1kHz**	**10-20kHz**	**16bit**	**96dB**	**1445kbps**	**10.8MB**
Sample rate	**22.05kHz**	**11.025kHz**	**16bit**	**96dB**	**722kbps**	**5.4MB**
Stereo to Mono	**22.05kHz**	**11.025kHz**	**16bit**	**96dB**	**361kbps**	**2.7MB**
Sample rate/size	**11kHz**	**5.5Hz**	**8bit**	**48dB**	**90kpbs**	**677kB**
Sample size Mono	**5.5kHz**	**2.75Hz**	**8bit**	**48dB**	**45kbps**	**338kB**

Figure 0-23

As shown in the table above, to achieve modem bandwidth transmission, the quality is below that of standard telephone conversations. Even at a 5.5kHz sample rate, the average datarate is 45kbps, above what a modem can handle. With some clever streaming, audio codecs have compression ratios of **~50:1**. The 5.5kHz clip could be compressed to an average datarate of 5-8kbps. The audio portion of the Intel Video Phone sustains an audio stream of about 5.3kbps. With "silence detection" enabled, this portion of the bandwidth can be given back for other tasks until a player speaks again.

Streaming audio technologies on the market today are

EchoCast, GamePhones™ SDK - Echospeech	***www.echospeech.com***
Internet Wave - VocalTec Ltd.	***www.vocaltech.com***
Shockwave Audio - Macromedia	***www.macromedia.com***
ToolVox Gold - Voxware Inc.	***www.voxware.com***

2.6.1.2.1 Audio residing on CD

Many game titles have pre-coded "audio taunts" within their game. However for multiplayer games, this trend did not continue. Sure, many multiplayer games provide text communication, but taking your hands away from the controls just to type something could get you killed! Instead, a series of different taunts could reside on the CDROM and could be played back on the local system when a remote opponent wants to taunt you. Much like video residing on a CDROM, invoking an audio taunt would only require a small command to be sent.

2.6.1.2.2 Game Logic with Audio and Video

These Internet media technologies have been introduced in the past year. Several of them have been adopted on various websites. They have successfully added more life to web pages, but it's not yet clear how well they will work with multiplayer games.

Having continuous communication is a plus, but when network latency rises and/or the game state requires more bandwidth, the extra bandwidth comes out of the audio/video portion. This issue can be addressed through the proper prioritizing of the different classes of data. Video takes up the most bandwidth and in most cases games can do without it temporarily. The next level of priority is more difficult to decide on. Depending on the type of game, one could argue that audio dropout occasionally is acceptable in order to keep the game running smoothly. However, if the game is a strategy game that requires continuous communications between teammates, and is not a high-action 3D game, then audio should be the highest priority.

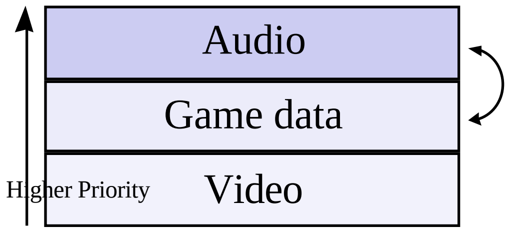


Figure 0-24: Media Hierarchy

To stretch this even further, we can correlate these schemes to the type of gaming model that can implement audio/video. A peer-to-peer model with a large number of connections may not be practical. A server-client model may require a powerful server to process the game and well as transfer the media data. The fully connected model makes the most sense in that the game data and media data can be partitioned. The server handles game data, with perhaps a little additional TCP/IP information sent to each client. The additional information that is passed to each client gives the clients the opportunity to "directly dial-up" other players and communicate to them. Each client is responsible for their own media communication with others, thus offloading this task from the server as in the server-client model.

2.6.2 Tips

TIPS FOR SCALING TO THE PLATFORM

- Cache information about other players locally
- Use local storage whenever possible
- Balance the CPU load

2.7 Golden Rule 7: You're Not Done Yet: Business Considerations

There are, of course, other issues involved in creating a successful multiplayer game. Many of them revolve around the business considerations. Just how do you intend on making money off of the title? Will it be via a subscription service? Pay-for-play on proprietary servers? Will you work through an Online Service Provider (OSP)? How will the billing work? What about security? How will users get updates to the game? Will you allow for an open architecture so that others can make modifications (new levels, etc.)? These are but a few of the issues involved on the business side (which is beyond the scope of this paper), but, nevertheless, they are all issues which must be considered for a successful title.

3. CONCLUSION

We are currently experiencing the birth of a new technology trend in multiplayer games. Although these new games have the ability to become a significant new category in the computer gaming market, the quality and availability of bandwidth hinders this forward progress. The Internet infrastructure is limiting insofar as it does not currently support many of the best ideas of the connected multiplayer gaming community. As some OSPs have found out, the best method of ensuring a long-distance, low-latency, high-bandwidth Internet connection is to architect and build your own network from point A to point B. This eliminates all unnecessary data packet misdirection and congestion. However, this is an expensive and therefore prohibitive solution for a game developer to rely upon.

We've discussed the Internet's latency and bandwidth limitations, the 'Golden Rules' we've outlined in this paper are an excellent start at understanding some of the issues involved in developing connected multiplayer games to work around these limitations. With the new crop of faster, more capable PCs and their accompanying OS', the gaming platform of today makes an attractive target.

The possibilities of online gaming are seemingly endless; the industry needs to have the foresight and imagination to test the boundaries, and the means to push this new technology forward.

Appendix A. References

Winsock
www.stardust.com
www.intel.com/ial/winsock2/

DirectPlay

www.microsoft.com/developer/tech/dx3doc/

RSVP
www.cisco.com/warp/public/724/4.html
www.intel.com/ial/rsvp

Kali
fgi.net/~jhoegl/noframes/kali.htm

Streaming Video
- ***www.iterated.com***
- ***www.vxtreme.com***
- ***www.vdo.com***
- ***www.xingtech.com***
- ***www.microsoft.com/netshow***
- ***www.real.com***
- ***www.vivo.com***

Streaming Audio
- ***www.vocaltech.com***
- ***www.macromedia.com***
- ***www.voxware.com***
- ***www.echospeech.com***

Mpath
www.mpath.com

Total Entertainment Network (TEN)
www.ten.net

America Online
www.aol.com

Game Developer Magazine Special Report ***Online Game Development*** Winter 1996/1997
www.gdmag.com/online2.htm

Web Techniques Magazine, March 1997
www.webtechniques.com

Jupiter Communications Online Gaming Report 1996

Intel's Developer Relations Group
www.intel.com/drg