June 2001

Broadcasting Virtual Games in the Internet

by Martin Otten (martino@valvesoftware.com)

Abstract

This paper provides an introduction to current possibilities for broadcasting an online game to a large audience via the Internet, especially for teamplay games taking place in a 3D virtual environment. Our primary focus is broadcasting technologies which are highly scalable and flexible. We show that a virtually unlimited number of spectators is also attainable using conventional unicast routing as well as multicast routing. The second focus of this research is an autonomous camera planning agent, capable of using information extracted from live game play to choose camera positions. These features have been successfully implemented in our HLTV proxy which collects, buffers, analyses, and broadcasts 'Half-Life' multiplayer games.

Introduction

Today the Internet has a steadily growing community of online game players. They take part in shared virtual 3D environments following rules given by the particular game they play. In the year 2000, the Interactive Digital Software Association (IDSA) made some surveys in the US and found:

- 60% of all Americans play video games, or about 145 million people.
- 35% of all Americans rated playing computer and video games as the most fun entertainment activity for the third consecutive year.
- 43% of people who play interactive games are women.
- The average age of an interactive game player is 28 years old.
- There were $6.0 billion in U.S. entertainment software sales in 2000.

Furthermore, online game playing has become a serious sport much like traditional sports which are enjoyed and celebrated by millions of people today. This is a result of the increased average Internet bandwidth and the high visual and audio quality of modern computer games. Computer players around the world are forming teams (often referred to as Clans) and playing against each other in different leagues. Such competitive online gaming is also known as 'cyber sport' or 'e-sport'. Gaming related organisations, some of which were founded in the early years of online gaming, are currently attempting to become leading national or international forces in the field. These organizations arrange professional international events offering prize money beyond the $250,000 mark. They are sponsored mainly by large hardware manufactures (especially CPU, video card and network). These organizations have a huge interest in doing this, since modern home PCs have attained performance levels which are not required by normal desktop applications. Currently, high-end 3D games

are one of the few applications capable of stressing today's PCs to their limits. In this respect, computer game players are the perfect target group for advertising new computer hardware. In addition, there exists another group of potential investors who are currently inactive in this space: TV stations and media groups. The visual quality and production level of computer games has reached near-Hollywood movie standards and media companies are beginning to understand that it could be profitable to produce TV content in a digital virtual environment.

In addition, the Korean World Cyber Games (WCG) is trying to make computer gaming part of the Olympic games. The WCG often stages large events with substantial monetary prizes. The WCG is working together with large entertainment hardware & software manufactures such Samsung and Sony. Currently, the International Olympic Committee (IOC) dos not yet appear interested in hosting computer games since definitions and rules are quite unclear. Yoosup Oh (CEO of International Cyber Marketing Co), when asked about his vision of the future of competitive computer gaming, commented:

"With the continuing development of data streaming technology, games will be brought to viewers very clearly, but rules need to be set to make the games look and feel like classic sports. In this way we can convey the tactical aspects of games to the average spectator, so he or she can enjoy the game and fully understand it. Game rules should be very simple and clear, because most sports fans don't really dwell on tactical issues. All that matters to them is whether the ball is in or not, so to speak."

Broadcasting computer games

But how do you watch an online game today ? The answer is: you can't. To date, none of the commercially successful computer games offer special options to give a large number of spectators access to watch the game. The most common way of "watching" is to meet in an Internet chat room (IRC), and waiting for text messages telling the current score to be automatically generated by server scripts. This experience offers a very limited excitement level. Additionally, after the game is completed, it's possible to download a recorded demo which would allow seeing the game through the eyes of a participating player. While this is an improvement over text messages, the method is still insufficient since only a small aspect of the game is shown. This doesn't work at all in team games. For example, imaging a football game through the eyes of a single player would make it very difficult to develop a feeling for the rules of the game.

Currently, there do exist approaches to broadcasting a game via Internet proxies. However, since these programs are not supported by the game industry, they are instable and complicated to use. They also don't provide any kind of serious camera and commentator management by a director. In most cases you simply see the view of a participating player. Since most of today's sport audiences are not active players themselves, they want a clear understanding of 'what's going on', possibly including a commentator explaining rules and commenting on team strategies. If you watch a game on TV, you expect to see only interesting moves and even a slow motion replay for the most important scenes. So, an average sports event on TV defines the exact

requirements for reaching a wide audience by broadcasting virtual games in the Internet.

First, we need computer games designed to be broadcast to huge audiences. In this respect, the game play must be simple. A spectator without knowledge of the game should be able to watch the game for 30 minutes and by that point should recognize the main goals and rules of the game. A game between two parties shouldn't be too long, right now a normal match has two rounds, 20 minutes each. The world (or level) design should be easy to follow. Lots of small rooms on different floors are not only confusing the spectator, they are also slowing down the game play itself. The world should be mostly flat in a relatively open environment. Most of the famous sport fields are a simple rectangles with only two goals or baskets. This allows the audience the see the whole game from a single viewpoint. Sure, in a virtual environment, we have much more possibilities to present a game, but we shouldn't move too far away from this proven concept. Also, the world and it's rules shouldn't change too often. This allows spectators to follow the game play more easily and possibly even predict what will happen next and speculate about team tactics.

Figure 1: Game overview with inset window

Our second requirement is an Internet framework that allows us to broadcast a semi-reliable data stream to an unlimited number of clients. Multicast transmissions (RFC 966 & RFC 1112) would be the first choice, since an optimal routing without data redundancy is provided by mechanisms located at the Internet routing level. But multicast isn't available to the broad public right now. Most ISPs don't offer multicast routing for their customers. The huge multicast research network (MBONE) is only

accessible for universities and involved companies. But multicast applications are becoming more important since multicast is needed for any high-bandwidth one-to-many multimedia application using for example audio and video data streams. The big router manufactures already responded to the huge demand for multicast and most hardware routers are now able to handle multicast streams. Multicast is also implemented in the core definition of IPv6 and will be a common supported Internet feature in the future.

Unfortunately, since multicast isn't broadly available right now, we had to pseudo-implement it on our own using conventional unicast IP routing as the actual base protocol. A single host can broadcast the game only to a limited number of connected spectator clients, depending on the host's hardware and network resources. But if we chain together many hosts broadcasting the same game, we are able to build a virtual tree of routers and therefore are able to make this system highly scalable.

Finally, we need an entertaining and informative view for spectators. Each spectator should be able to choose for himself how active or passive he wants to be while watching a game. A completely passive spectator just wants to lean back and enjoy the game without any further interactions. He assumes he will get a perfect show, seeing all important events from interesting viewpoints. At all times, he expects to have a clear understanding of the current situation provided by additional comments and visual information (scores, time, inset map, etc). A more active spectator, on the other hand, would choose the camera views on his own, might want to move freely throughout the virtual environment, and might want to track his favourite players or browse complex game statistics. Both kinds of spectators should be supported, even if the active and more sophisticated spectator will initially be a minority.

Providing the 'perfect' view for passive spectators in real-time or with a minimal delay is a complex problem. But not only for virtual environments, it's also an old problem at traditional live sport events. Most of TV content is highly scripted, so each actor, camera man or director knows exactly want will happen in the next seconds. This fact makes it easy to position camera and lights correctly even before the action starts. But sports are inherently unpredictable yes still must be broadcast nearly live. The fact that nobody on the whole wide world knows how a match will end until it's over is one of the main reasons of the success for sport as entertainment. The tension of the event is lost if everybody knows the final results because they have been watching the game with a 30 minutes delay. Thus, the capability to broadcast live (or nearly live) is also an absolute requirement.

The common setup for broadcasting sporting events is a production truck containing a mobile video editing studio. The director and commentator can see simultaneously all camera views and can decide which camera is 'on air'. The director coordinates the camera crew and inserts additional replays or effects (slow-motion etc). This idea has already been tested in various virtual environments and has worked well with an experienced team. But nevertheless this solution requires lots of well-trained man power, at least 6 people (one director, one commentator, and at least 4 cameramen). In our virtual world, the cameramen is unnecessary since we are in a virtual world and all cameras could be set by the director with a single click. But like in the real world, cameramen are actually intelligent agents, filtering important information for the director. Since they have knowledge about the world they're filming, they can act

autonomously, predicting where interesting events might occur in the future and also adding some artistic value by choosing camera positions that emphasize the current situation.

Our hypothesis is the idea that several autonomous camera agents and one director who decides the final view can be implemented by intelligent program code, knowing the rules of the game and following general cinematic guidelines. If the game stream is buffered for a short time (about 30 seconds), the agents would even know what will happen in the near future. This is a big advantage over traditional television production requirements because no human intentionality is needed to predict the game. An artificial director agent could even be supported by a human director if needed. Research about camera planning in virtual environments has already shown useable results.

Related work

Broadcasting computer games is an unexplored topic since necessary Internet bandwidth has only become available only a few years ago. The online community around the first person shooter 'Quake' by id software took advantage of id's liberal source code release policies. Id actively offered parts of the game source as free downloads encouraging people to alter game content and create completely new games. Some programmers began to reverse engineer the network and demo file format and created new tools (LMPC, KEYGRIP) to edit demo files. These tools are very similar to professional video editing tools. Making small movies within a game engine (called Machinima) has become quite popular. Others used the knowledge about network code to give themselves advantages during gameplay. They built small 'cheat' proxies (Qizmo, Cheapo), that hooked into the game data stream between server and client. These proxies could insert additional information about other players and some could even aim and shoot automatically for the player.
'Quake' also allowed players to enter a spectator mode where they could fly through the level and watch the game (but were invisible to other players). Unfortunately, each such spectator occupied a valuable normal player slot and most servers could not offer more than 32 total slots. So the proxy technology was used to offer the view of one spectator for other clients that were connected to the proxy, not the server. These tools (QTV and GTV) met with some success, but they also suffered from several large problems. First, they were not programmed by the original game company and were therefore very fragile with respect to changes that company might make to the games. Also, they didn't offer any kind of directory service to find currently broadcasted games. Spectators had to know time and IP address to watch a certain game. Furthermore, these proxies had no camera planning and all users saw through the eyes of a 'camera' spectator on the game server. If this cameraman missed an important event, spectators would also miss that event.

Camera planning in a virtual environment is actually a fairly well researched scientific field. The 'Virtual Cinematographer' gives a good introduction in general principles of cinematography. They encoded cinematographic expertise as a hierarchically organized finite state machine. The 'UCAM' project created a camera system that automatically follows users activities, providing an external view of the current scene.

The system could learn the users favourite directorial style, adjusted viewpoint style, transition pace and style as preferred by the user (User-Sensitive Realtime Camera Planning). Camera planning could then be expressed as a set of constraints and the best shots would be evaludation using a constraint solver as shown in 'Model for Constraint-Based Camera Planning'.

How to produce a TV show within a virtual environment, including coordination of the production team is shown in 'Creating a Live Broadcast from a Virtual Environment'. In that paper, a game show with actors controlled by human players was broadcast to an audience by humans assuming traditional roles such as director, cameraman and video tape operator.

HLTV Network Architecture

We had several design goals when starting the HLTV project. First of all, the server should be virtually unaffected by the new technology. The game itself is most important and, therefore, we couldn't afford to stress the server process more than needed. Implementing this feature into the server would have mean less CPU & network resources for the players and other more critical code and that effect would negatively influence the integrity of the game world. So we decided to create a separate application that completely mirrors the game and its history in a second process space. Thus we can work on the world data remotely without disturbing the running match. This application is the HLTV master proxy that is handled by the server like a normal spectator client with one exception. The master proxy gets all data about all players, items and game events, not just those located in the surrounding area (PVS – Potential Visible Set). This culling of information is done for normal clients to reduce bandwidth usage, but the master proxy in fact wishes to capture the complete game so that it can later show any event or object from any viewpoint. In the final implementation, the only additional load on the server was a single new client slot and changes to the server code were kept to a minimum.
A virtual environment with fast moving and highly interacting objects produces a large quantity of data: consider, for example, in a world comprised of 512 items where at least 32 of them are changing quickly direction and position (about 20 times per second). Saving this much 'game state' as plain data would waste about 1 MB memory per second. Thus we are stored the complete game data in a compressed format, using the 'delta' compression (also called PICA – Protocol Independed Compression Algorithm). PICA only transmits changes relative to the last mutual agreed state. This way the memory requirement is lowered to 2 MB per minute, which enables us to record a complete 30-minute game and support random access to all of that data on a traditional PC.

Buffering the game has some additional benefits. By delaying the game, we reduce the probability to use this spectator software as 'cheat' for the participating players (because they would know all enemy locations and their status). Fortunately, players are moving quite fast within their virtual environment, so this information losses its value after a short amount of time. The chosen delay depends on the general game pace. In general, 30 seconds is a fair trade-off between cheat protection and near-live transmission. Another reason for buffering the data is to allow for more sophisticated

camera planning logic to execute. These algorithms take a great advantage by knowing the near future, so the camera could retroactively be placed in the appropriate position even before the particular event in the scene starts. Larger game buffers up to several minutes are possible if a human director wants to insert replays of important game events.

HLTV Network design

The network design offers two different ways to broadcast a game. By using the multicast technology (Figure 2), the problem of optimal packet routing to a group of clients without data redundancy is completely solved by the network routing layer. To join or leave such a multicast group, we are using IGMP (Internet Group Management Protocol), which is supported by all Berkeley Socket compatible libraries.

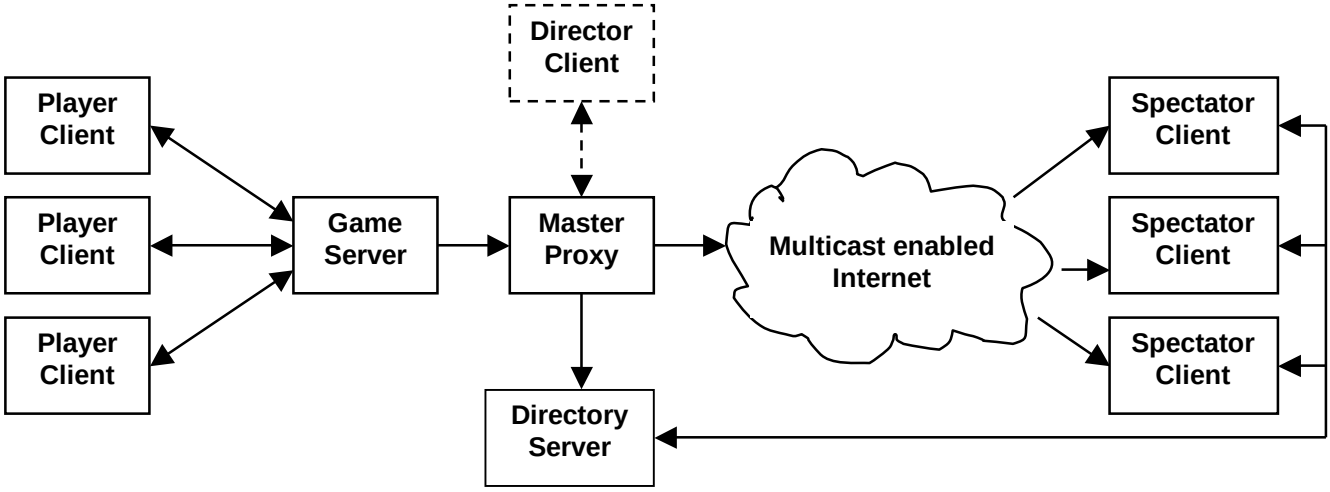

Figure 2: Multicast Architecture

The proxy sends data into two different multicast groups. The first group provides signon data (necessary start-up data to load correct game type, level and player models). Since we don't know when a client joins the signon group to watch a game, we have to resend the signon data stream every few seconds again. Once a client received all of the signon data, it then leaves the first group and joins the second group that broadcasts the actual game data. All clients stay in this group until the server changes the game level or type.

The game data stream is also delta-compressed to save required network bandwidth. Delta-compression assumes that all recently send packets were properly received. Thus a single lost packet could completely interrupt the game stream decoding, since the client cannot simply send a request for a resend. Because multicast doesn't provide reliable transmissions (yet), we have to broadcast at a fairly frequent interval

an uncompressed game state to give clients a chance to recover from packet loss. This broadcasting of redundant data to correct transmission errors is called Forward Error Correction (FEC).

If multicast routing is not enabled, additional HLTV proxies can be set up to create a tree of relay proxies (Figure 3). As currently supported by HLTV, each relay proxy may serve up to 128 clients which could be spectators or other relay proxies. Relay proxies are the same application as the master proxy, but they don't purposely delay the game stream or analyse it to add camera commands. They only need a small packet buffer used by the delta compression. Each relay proxy registers itself at a general directory server (Master Server), following a special naming convention, thus the Master Server can assign all proxies broadcasting the same game to a unique group. If a user browses the game list given by the directory server, all proxies broadcasting the same game appear as a single proxy. During the connection process the client application evaluates the closest/fastest relay proxy to itself.

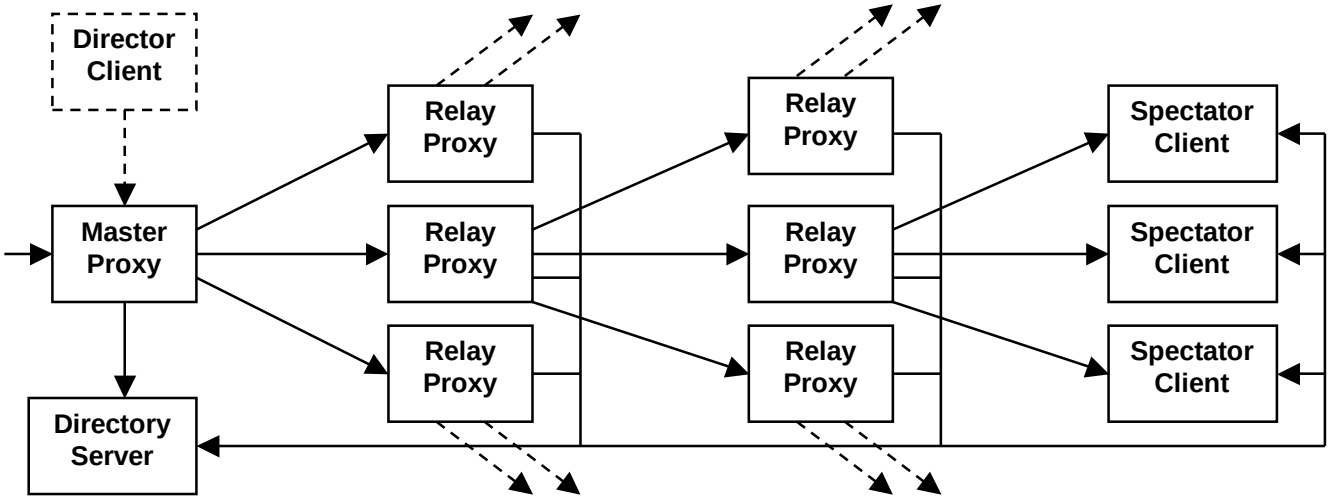

Figure 3: Unicast Relay Architecture

Since multicast technology and the tree of relay proxies don't interfere, they could be used simultaneously. The master proxy can broadcast game data into a multicast group and serve relay proxies at the same time. By using this method, all connecting clients would first try to join the true multicast stream. If that fails (because at least one router between client and master proxy doesn't support multicast), the client falls back to the best relay proxy connection. This attempt and fallback are transparent to the user; the user simply clicks on a desired game.

HLTV Application Architecture

The master/relay proxy is the HLTV core application (Figure 4). It's mainly programmed in object-orientated C++, with some helper functions and time critical code written in functional C-style code. All modules are part of a main framework

that registers, initialises and executes all subsystems in a Round-Robin order. New modules with new commands can be easily added. The network module is a special module that can run in it's own thread to reduce network latencies. Thread-safe communication between modules is guaranteed via FIFO queues protected by semaphores.

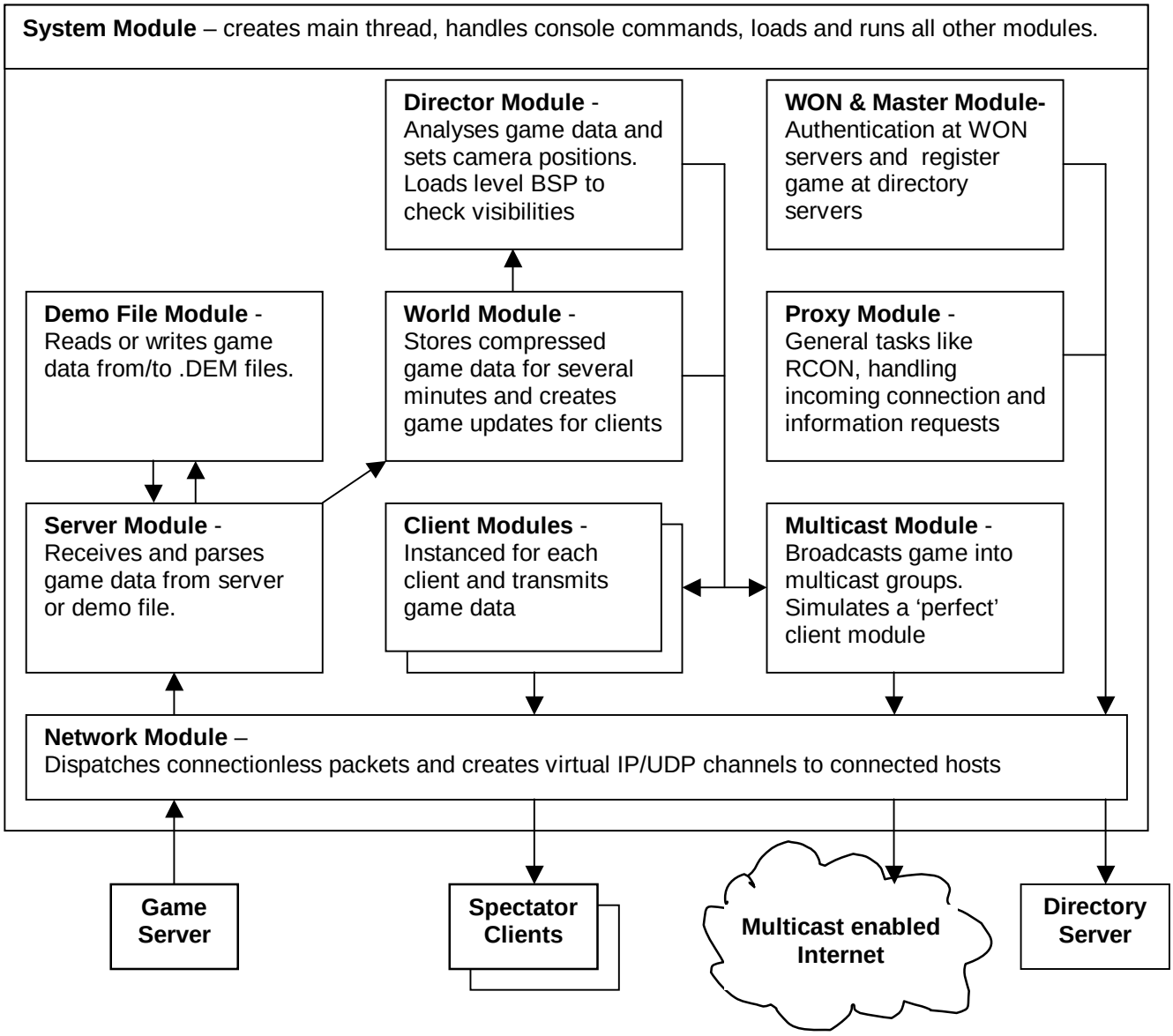

Figure 4: HLTV Proxy Design

A good way to understand the different modules and their relationships is to follow the game data flow. First the Server Module connects to a game server, parses all sign-on data, and initialises the same level with the same setting in the World Module. After the connection is complete, the Server Module receives about 20 update packets per seconds. Each world update is relayed to the World Module that saves all world updates in a storage buffer. This game buffer is compressed and ordered by time. By default this buffer takes up to 240 updates, that equals about two minutes of game play. Since the buffer is compressed, 120 updates (one minute) requires about 2 megabytes of RAM (instead of 8 MB uncompressed) . Thus it's quite easy for a modern PC to buffer a whole 30 minutes match.

The Director Module waits until enough data is available in the game buffer(by default 10 seconds), then it analyses that data and calculates the best camera view for this scene. Special event messages generated by the game logic running on the game server are intercepted by the Director Module. They tell the Director Module that an interesting event occurred and what objects are involved in that event. The game logic can also provide further information about the character of the event (dramatic or more informational, etc). That information helps the director logic choose camera positions and angles that emphasize the scene in a compelling fashion. For example, to stress a dramatic situation the Director Module chooses a lower and closer camera position towards the players. The director logic doesn't have to know the actual gameplay mechanics. All it knows is that the world is a 3D environment with interacting objects.
If no information about game events is available (maybe because nothing happened or the game type doesn't support HLTV), the Director Module tries to find positions, where as many players as possible are visible. Therefore it considers all of the level objects to determine a players' visibility. The logic also uses object directions to identify interacting objects (players/objects facing directly towards each other are most often considered to be involved in some kind of relationship).

For each connecting client (spectator, relay proxy, or director) a new Client Module is spawned and added as a new system module. Each Client Module creates a virtual UDP channel to its client host via the Network Module and reacts individually to the client's network bandwidth (rate) and packet loss. The Client Modules continuously ask the World Module for an actual update for the current client time. The master proxy's World Module handles three different timelines: server, client and director time. The world time is synchronized to the game server's last update. The client time is the point of time all clients currently see. This time is always before the server time, the concrete delay can be set by the proxy operator with the "delay" command or by a connected director. To avoid cheating the default value is 30 seconds. Client time doesn't have to follow the server time in a linear way. It can be stopped to freeze the game, it can be decelerated to create a slow motion effect. It can even go reverse, backwards in time. All these effects are easily added by the director to replay or emphasize particular events. All spectators have the same client time, while only the director can alter his time as he wants to. All times must be within the proxy's game buffer range. (It was quite difficult to handle and synchronize all these times since there is a fourth client time needed to realize jumps back in time for clients.)

The Multicast Module is very similar to a client module, except that it is created once at startup. This module blindly sends data into two different multicast groups (signon and game data), hoping the data is received correctly by joined multicast spectators. It uses the same virtual UDP channels offered by the Network Module, but it creates fake incoming packets that acknowledge every outgoing packet.

The WON & Master Module authenticates the proxy at the WON servers as valid host and registers itself at the game directory servers (Master servers). This way users will find the proxy while searching for current games.

Future Work

Unicast as well as multicast network implementations are nearly finished. We have already created larger test environments with game servers in north America and a HLTV proxy chain broadcasting the game to Europe. Also multicast transmission test into IGMP enabled LANs were successful.

The latest HLTV proxy version also supports a commentator voice stream. This 'Voice-Over-IP' feature is using a low bandwidth voice compression that's suitable even for 56KB modem users. Since the game and voice data are transmitted via the same network stream, game and commentator speech stay synchronized even if they are delayed or buffered by HLTV relay proxies. This is a big advantage over using a separate 'Voice-Over-IP' product like Shoutcast or BattleCom.

The main challenge of future improvements will be to create more intelligent director logic and camera management logic. Currently, these systems are very limited. A better solution would be a formal language that would be able to describe common cinematographic knowledge. Based on this language we could set up rules used by the director logic to make decisions about the right camera set up and cut timing. These rules should be defined as general as possible to be independent of the particular game type. The camera management logic should offer lots of possible shots, to allow a broad variety of ways to emphasize character of a scene. Right know the camera just follows the primary actor and tries to keep the secondary object/actor in view. We need static cameras that show the scene as a whole and we need zooming/moving/sliding cameras.

Another large undertaking will be the director application that enables one user to choose any camera angle, anywhere and at any time during a running game. This director would closely work together with the voice commentator to insert replays, images (even possibly advertisements), or any other effects. From our experiences, we do not believe that the director and the commentator can be the same person, since commenting a game requires a lot of concentration which would preclude most other tasks. The director tool will have much in common with classic video editing applications. Thus, this will be a more of a technical programmer job than a scientific problem.

Related Internet Resources:

http://www.gamespy.com/stats
http://www.idsa.com/pressroom.html
http://www.thecpl.com/
http://www.worldcybergames.org/

Videos showing HLTV:

http://www.slipgate.de/download/cb-hltv1.avi
http://www.slipgate.de/download/cb-hltv2.avi