HEAVEn: The Design and Implementation of a Scalable ATM Based Distributed Virtual Environment

Prepared by
Kevin Stuart Dennis

Prepared for
Mr. M. J. Ventura

Department of Electrical Engineering
University of Cape Town

This dissertation has been submitted to the Department of Electrical and Electronic Engineering in fulfilment of the academic requirement for the Degree of Master of Science in Electrical Engineering.

February 2003

Acknowledgements

I wish to express my sincere gratitude to the following people for directly and indirectly assisting in the preparation and completion of this dissertation:

- Mr. M. J. Ventura, my supervisor, for his guidance throughout my time in his research group.
- Stuart Doyle, my dissertation partner, for his input and collaboration.
- My colleagues in the Communications Research Group in the Department of Electrical Engineering at the University of Cape Town. Their constructive advice and friendship made my time in the group a memorable experience.
- The authors of the many papers that were read and referenced during this dissertation.
- My "proof-readers," Dianne King, Steeve Chung, Nelia de Abreu, Sven Shepstone, Neco Ventura and my father, mother and sister, Michael, Bessie and Lee-Ann Dennis, all who put time and effort into reading through this dissertation and aiding me in removing the multitude of grammatical and spelling errors.
- My friends, especially Philippa Slack, who made that extra special effort to keep me smiling and who continually reminded me that there is life after my dissertation!
- My parents and sister, for their support, their care and the time spent helping me in every way imaginable.
- My Lord and Father, Jesus Christ, for comfort and assurance despite the adversities that hampered me throughout the time spent working on this dissertation.

Declaration

I declare that this dissertation consists of original material. Where collaboration with other parties has occurred or material generated by other researchers is included, the parties and material are appropriately referenced.

This dissertation is being submitted for the Degree of Master of Science in Electrical Engineering at the University of Cape Town. It has not been submitted for any degree or examination at any other University.

signature removed

Kevin Stuart Dennis
17 February, 2003, Cape Town, South Africa

Synopsis

Over the past few years, the rapid advances in 3D graphical technologies have solved, for the most part, the problems associated with rendering complex or realistic scenes in real-time on desktop computers. These capabilities have not yet been fully utilised in the creation of virtual worlds or virtual environments.

A multi-user, scalable, interactive, networked virtual environment is a tool that allows many users in geographically remote locations to interact within a virtual environment in real-time. The environment can represent many situations, for example: a board meeting for an international company, a class of doctors where the lecturer is demonstrating on a 'virtual cadaver' or a simple chat room that people can meet in and where they can socialise.

Virtual environments have, in the past, been designed for a specific purpose such as the many military simulations. These virtual environments often require dedicated networks to function properly and even with a dedicated network, require extensive hardware to support them. The design of generically and publicly usable virtual environments has, for the most part, been ignored due to lack of funding and the limited bandwidths available to clients. With the advances in 'last-hop' technologies, many of these limitations have been removed, allowing users the option of experiencing networked virtual environments from the comfort of their homes.

In this dissertation, the design of the HEAVEn distributed virtual environment architecture is proposed. The aim of the HEAVEn design is to put forward an infrastructure that, when fully implemented, will enable users to connect to a generic virtual environment that guarantees the responsiveness of the environment to user interactions. The design proposes a framework of maximum delays that both the application and the networking elements of the design must adhere to if the responsiveness of the overall system is to be guaranteed.

The design offers scalability and ensures the consistency and realism of the environment. The HEAVEn design does this by using a client-server model that naturally limits the problems associated with the common peer-to-peer distributed virtual environments although this can limit the number of concurrently connected users. To prevent this situation and the bottlenecks that a single server can cause, the server is distributed.

The designed environment servers mediate various inter-user communications and interactions ensuring security and consistency of the environment while distributing the control across multiple environment servers to ensure the scalability of the design. Environment changes occur dynamically across all the interconnected servers ensuring environment consistency and persistency despite server or client failure.

While the design can be used in conjunction with any number of networking protocols, the design is specifically tailored for use in conjunction with an ATM and using native ATM connections. The ATM connections offer a quality of service that can guarantee the RT nature of the traffic between nodes in the designed environment. These guarantees are essential to ensure that the end-to-end application level response times between user action and response, as defined in the design, are met.

Another useful feature of ATM is the connection-oriented nature of its communication. This ensures that delivered data is already ordered and does not need to be buffered and reordered before use. This feature is immensely useful, as any extra delays in the data decreases the responsiveness of the virtual environment, as seen by the client. The size of an ATM cell is also an important factor as environment updates are often small enough to fit into the payload of a single ATM cell. The cell size is thus ideally suited to the transfer of environment updates. Yet another advantage of ATM is the availability of multicast connections. These connections join nodes requiring the same data but limit the network congestion that could be caused if the data was simply duplicated to all the destinations. This is done by only duplicating cells when necessary in the network. While multicast technology was originally thought to be the focus of the design, it was found that the multicast technologies were not ideally suited to the transfer of most of the environment data although the technology may still be useful in certain cases as described in the design. The provision of these services by the network removes the need for intermediate technologies that would provide similar services to the applications and this further reduces delays in the overall responsiveness of the environment to user interactions.

The HEAVEn design differs from other virtual environment designs in that it separates the data being passed between clients and servers into temporal and delivery critical data. These two streams are processed separately to enable their specific needs to be met. The design includes several other features including the option of balancing server load, hiding the environment server complexity from the user, methods of collecting statistical information from user interactions and means to limit the quantity of data being sent and received by the clients and servers within the virtual environment.

In order to prove the design's ability to support a virtual environment, the HEAVEn design is implemented and a test-bed constructed in the Communications Research Group at the University of Cape Town. This test-bed will be used for future evaluation and optimisation of the architecture, and in research aimed at defining the network characteristics of distributed virtual environments based on ATM technologies.

Conclusions and recommendations for future work are presented. It is concluded that ATM suits the design requirements for a distributed virtual environment due to its reliability and guaranteed service offered. Moreover, the HEAVEn design is a scalable, networked virtual environment that supports RT interaction with the users of the environment. It is thus recommended that further work be done to implement the design on a real-time operating system to further guarantee the response times to the users' interactions.

Contents

List of Illustrations

List of Tables

List of Abbreviations and Acronyms

Abbreviation	Explanation
2D	Two Dimensional (Flat screens are two dimensional)
3D	Three Dimensional (The Real World around us is 3D)
AAL	ATM Adaptation Layer
ABR	Available Bit Rate
ADSL	Asymmetric Digital Subscriber Line
ANS	ATM Name System
AoI	Area of Interest
API	Applications Programming Interface
ARPANET	Advanced Research Project Agency's Network
ATM	Asynchronous Transfer Mode
BiP	Break in Presence
B-ISDN	Broadband-ISDN
bps	Bits Per Sample or Bits Per Second (both standard acronyms in different fields)
Bps	Bytes per Second
Broker	Broker Server (*plural*: Brokers – Broker Servers)
BS	Broadcast Server (*plural*: BSs - Broadcast Servers)
CAVE	CAVE Automatic Virtual Environment
CBR	Constant Bit Rate
CD	Compact Disc
CLP	Cell Loss Priority
CRC	Cyclic Redundancy Check
CS	Control Server (*plural*: CSs - Control Servers)
DARPA	Defence Advanced Research Projects Agency
DIS	Distributed Interactive Simulation
DIVE	Distributed Interactive Virtual Environment
DNS	Domain Name System
DOF	Degrees of Freedom
DSL	Digital Subscriber Line
DVE	Distributed Virtual Environment (*plural*: DVEs – Distributed Virtual Environments)

Abbreviation	Explanation
FAST	Frame Based ATM over SONET
FCFS	First-Come-First-Served (Similar to - FIFO)
FIFO	First-In-First-Out
FoI	Field of Interest
fps	Frames Per Second
FTTC	Fibre-to-the-Curb
Gbps	Giga Bits Per Second
GFC	Generic Flow Control
GFR	Guaranteed Frame Rate
HEAVEn	HEAVEn is an Experimental ATM-based Virtual Environment
HEC	Header Error Check
HFC	Hybrid Fibre Coax
HLA	High Level Architecture
HMD	Head-Mounted Display (*plural*: HMDs – Head-Mounted Displays)
I/O	Input/Output
ID	Identification
IETF	Internet Engineering Task Force
IP	Internet Protocol
ISDN	Integrated Services Digital Network
ISP	Internet Service Provider
Kbps	Kilo Bits per Second
LAN	Local Area Network (*plural:* LANs – Local Area Networks)
LoD	Level of Detail
MAC	Media Access Control
MAN	Metropolitan Area Network
Mbps	Mega Bits per Second
MCR	Minimum Cell Rate
MPLS	Multi-Protocol Label Switching
MS	Model and Media Server (*plural*: MSs – Model and Media Servers)
MUD	Multi-User Dungeon
MUSH	Multi-User Shared Hallucination
NIC	Network Interface Card (*plural*: NICs – Network Interface Cards)

Abbreviation	Explanation
nrt-VBR	Non-Real-Time VBR
NS	Name Server (*plural*: NSs – Name Servers)
OC-n	Optical Carrier level n
OO	Object Orientation or Object Oriented
OS	Operating System
PC	Personal Computer (*plural:* PCs – Personal Computers)
PDU	Protocol Data Unit (*plural:* PDUs – Protocol Data Units)
PT	Payload Type
PVC	Permanent Virtual Circuit
QoS	Quality of Service
RAM	Random Access Memory
RT	Real-Time
rt-VBR	Real-Time VBR
SIMNET	Simulator Network
SONET	Synchronous Optical Network
sps	Samples Per Second
SVC	Switched Virtual Circuit
TCP	Transmission Control Protocol
UBR	Unspecified Bit Rate
UDP	User Datagram Protocol
UNI	User-Network-Interface
VBR	Variable Bit Rate
VCI	Virtual Circuit Identifier
VDSL	Very-High-Data-Rate DSL
VE	Virtual Environment (*plural*: VEs – Virtual Environments)
VPI	Virtual Path Identifier
VR	Virtual Reality
WAN	Wide Area Network
WTK	Sense8's WorldToolKit™

Chapter 1 **Introduction**

"The field of Virtual Reality has been moving gradually into a new phase. From roots in the field of computer simulation systems such as flight simulators, and in science fiction, Virtual Reality is becoming a new and unique area in the world of computing." – David Stampe

1.1 What is "Virtual Reality"

"Virtual Reality" (VR) is one of the catch phrases often used in modern computing circles. It seems that everyone has his or her own ideas about what VR really is, what it does and how useful it is or will be at some future point in time.

The term 'virtual' is often used to refer to something that is being simulated. Personal computers use 'virtual memory' that is actually simulated memory stored on a disk drive, or a 'virtual disk drive' or 'RAM disk' that simulates a disk drive in memory. One of the most popular examples of a 'virtual' arena is the *Holodeck* in the *Startrek* [Par02] series. The holodeck is able to simulate a complete world in which the crewmembers can interact with both the simulation and each other. SGI's *Flight and Dogfight* [Sil02] were 3D flight simulators that inspired many engineers to develop networked virtual environments as early as 1984. Games such as *Wolfenstein 3D*, *Doom* and *Quake* [IdS02], where gamers can compete in a 'virtual' arena to battle the forces of evil, took the user even further and truly made 3D gaming a popular pastime.

'Reality' is what we perceive through our senses: sight, hearing, touch, smell and taste. A virtual environment (VE) is a simulated world where these senses are simulated in such a way that the user believes the simulation and experiences the simulation as if it was real. While current simulations are not yet able to fool our sensory systems completely, a virtual environment can portray a 'believable,' or 'real enough' environment to allow users to accept it and work within it.

'Believability' is a concept that might not be too familiar but if one considers movie-goers, they will often be particularly frightened or surprised by actions on a screen that, while only two dimensional pictures, are realistic and thus believable despite the screen, surroundings and other distractions. This is because the audience has voluntarily 'suspended their belief' in the reality around them and 'allowed' their minds to believe what they are seeing and hearing as they become involved in the movie. The same thing is true for a good book and even more true for a virtual environment where the

distractions can be minimised and where not only the senses of sight and sound can be simulated, but also touch and even taste and smell.

Original VEs were simulations displayed by a computer that a user could visualise, interact with and manipulate. Most modern descriptions of a VE define the term to mean a computer generated 3D model where users entering the VE can interact with the generated model and any other users within the environment itself. The model, known as the *environment*, does not need to be based on any known physical model and can literally take on aspects of our imagination.

Examples of VEs range from the graphically intensive interactive computer games based on outlandish worlds where aliens shoot biological weapons, to the highly precise military simulations where each piece of equipment is simulated exactly and where military personnel can be tested in situations that cannot be easily and safely recreated on a battlefield. Even a simple text based online Multi-User Shared Hallucination (MUSH)[1] can be classified as a VE and these MUSHs were in fact the first truly interactive VEs on the internet.

Virtual Reality is the experience of a VE in such a way that the user believes in it to such an extent that they are able to interact with it. Similarly, a Distributed Virtual Environment (DVE) allows multiple users to interact in Real-Time (RT) through a distributed VE architecture, even though the users may be located all over the world.

1.2 Why VEs have become popular

The many advances in technology and the competitive nature of the computing business have resulted in the rapid decrease in the cost of computer systems. These advances, coupled with the increase in speed and bandwidth of access networks, have offered connectivity to the masses and have resulted in the rapid growth of the number of users accessing the Internet. This interconnectivity has created the need for more interactive applications that users can use to communicate with each other across the globe. The question the users now ask is, "Why settle for 2D applications when 3D applications have so much more potential and appeal?"

[1] A MUSH or MUD is a text based multi-user online environment. Players from all over the world have combined to build these worlds and they are clearly the forerunners of the graphical interactive DVE games that are so popular today.

VEs have been around for some time, initially being only on single computer systems for use in 3D simulations, but later expanding to the Local Area Network (LAN). These VEs were very limited in their functionality and flexibility but did represent a technology that had potential. This potential seemed boundless but was initially limited by the VEs lack of usability and extensive hardware and networking requirements.

This potential is clearly visible in the modern VE applications that allow users to interact with computer generated 3D environment models. Such applications, often DVEs, are the distributed multi-user versions of the older VEs. The modern VEs allow users in geographically remote locations to interact with other concurrently connected users. The interaction takes place in real-time and within the confines of the environment model giving the users the impression that they are together in the same physical location.

These DVEs aim to create a sense of realism for their users by supporting realistic 3D graphics, sound, streaming media and other forms of information interchange that still remains realistic despite the distances between the users. DVEs are increasingly being used to aid in the creation of realistic environments in which communal tasks can be completed. DVEs have been found to be of use in many areas such as military training [Mac02], collaborative design used by *Boeing* [Boe02] and *Caterpillar* [Cat02], scientific simulation used by the Human Genome Project [HGP02], pain-relieving VEs [Hof02], the simulations used by doctors [Nat01] to map the human body and the ever popular games [Ash02] [Eve02] created by the entertainment industry. VEs have also been found to be useful tools for commercial shopping, remote control and sensing, communication, and much more. Further examples of the many uses of VEs can be found in Appendix E.

1.3 Distinguishing Features of a VE

The idea of a VE and a DVE has been highly developed since its initial conception in the 1950s and 1960s, but the general ideology remains consistent: a user enters the VE and interacts with it as if it were a real world.

Users connect to an environment using a console or workstation that must aim to provide the user with a sense of believable realism. It does this using various tools such as 3D graphics, surround sound, tracking and feedback. If the user believes in the VE and is able to interact with the environment, then

they are considered *immersed* in the environment and react to the stimuli within it as if it were real. This *immersion* is also known as a *sense of presence* [Str00].

An environment is noted for several features that enhance the realism of the environment. These features are needed if the environment is to remain realistic and useable. Any design needs to ensure that these features are foremost in the design's requirements. The main features are space, presence, time, communication and manipulation.

The users of a VE must share the same **space** or they will not be able to interact with each other. The space does not need to be a representation of a real place, but it must be *consistent* between the users or it will not be believable. The VE does not need to be represented graphically but the VEs that are graphical tend to be the most immersive due to the combination of sensory information that the user is offered.

The users in a VE can also alter the environment space and these alterations need to be stored, as the user will expect the changes to remain in the environment once they have made them. This is known as *persistency of space* and requires that the overall environment be *reliable*.

For the user to have a sense of **presence** within the VE, the user must feel that they are actually in the environment and not merely a spectator watching what is happening. The users must also be able to see each other. To do this, the user is represented using a model called an avatar. The user can therefore manipulate and interact with the environment through their avatar.

The avatars in the environment do not all need to be human controlled, leaving the option for some to be controlled by artificial intelligent engines or finite state machines. Tools and data sources can also be represented using simple models or avatars that are more complex.

The users need to be able to collaborate and to enable this, they need to have their actions completed on **time**. Without *synchronisation*, any collaboration would not be possible, as the users would not see the environment in RT.

The environment must also be able to respond predictably or the realism of the environment will be lost. This loss is known as a *Break in Presence* (BiP). If the user is expecting a certain response and it does not occur or occurs late, it will detrimentally affect the realism of the environment.

Studies [Wlo95] [Str00] have shown that while it is possible to tolerate a delay of between 100ms and a full second, the user will notice the delay but will still maintain their concentration although they might begin to feel disorientated or even nauseous [Har00], especially when wearing a head mounted display. The ideal maximum response time is 100ms for a fully immersive VE and 30ms for an augmented reality[2].

To ensure the realism of the VE, a maximum end-to-end delay must be stipulated in the architecture to be designed. The delays include both software and hardware components of the DVE.

Communication is imperative for interaction. This communication can take place via typed text, voice or gesture. The most effective and immersive VEs tend to have communication in all these forms, with the use of a Whiteboard[3] being the example of text-based communication. Users can write on the whiteboard while other users read from it or add to it if they have an appropriate writing tool.

The previous properties are all important to the function of a VE but they could also form part of a list of features needed for enhancing the design of a collaborative conferencing tool. The true usefulness of a VE lies in an environment's ability to allow users to share objects and even to **manipulate** the environment itself. This could take the form of moving objects within the environment or even altering the environment itself.

In summary, these five features separate the VE from common applications and allow users to communicate, manipulate objects and the environment itself and interact with each other [Sin99]. They are the cornerstones around which a design must be built. While there is a certain leeway associated with each of the features, the design must be able to support these features so that it can offer the users a believable and usable environment.

While it would be useful to stipulate values for the above features, it is important to remember that the features will be environment specific. For example, a gaming world could require high synchronisation to ensure an accurate sense of space while a conference world will require functionality aimed more at group communication. In these and other cases, the architecture to support

---

[2] An Augmented Reality is a virtual *Heads Up Display* (HUD) used to transmit data through semi-transparent screens.

[3] A whiteboard is a board that one can write on and view. The virtual whiteboards in a DVE work on the same principles.

the VEs will need to be able to offer all these characteristics and be able to ensure that the environment's specific needs are met.

It is relatively simple to guarantee responsiveness and achieve interaction with an environment on a single computer as the delays between the user's actions and the computer's responses are almost instantaneous and do not vary much. True 'interaction' must however include other users and this introduces a communications network, which in turn introduces several problem areas that must be dealt with when designing a DVE, or they could affect the realism of the users' experience.

Response times can be guaranteed for a single user system as the user's workstation has minimal delays that vary little. In a networked situation, communication times can no longer be taken for granted as the extra and random delays caused by congestion, propagation time and delays at network nodes can easily have an affect on the believability of the VE.

The environment must also remain **consistent** whether users join or leave the environment or portions of the environment's network fails. To ensure this, the environment must contain nodes that are able to reliably store information about the entire VE and be available to distribute this information to users within the environment. This enforces the need for persistent peers or servers that are able to distribute data throughout the environment. The fact that the users do not have all the data about the environment will force the clients to retrieve the information across the network and this introduces delays.

The network is not always a reliable transport mechanism and often errors can be introduced into data sent across communications networks, or the data could be lost entirely. The VE will need to be able to recover from the data corruption or loss and ensure that the persistency of space, a requirement of a VE, is met.

These problems need to be adequately addressed when designing the VE architecture and can be resolved using various algorithms for correcting errors, retransmitting lost data and predicting or assuming data that is lost or delayed.

It is also important to realise that if the VE is to be a tool that is useful and that many users will want to use, then the architecture supporting the VE needs to be scalable to allow for an increasing number of

users. **Scalability** must be designed into the architecture to allow simple expansion, and efficient use of the resources used to store, communicate with and update the environment.

Both the features and the problems need to be addressed when considering an architecture to support a networked VE. Unfortunately, the needs of the VE are interlinked in such a way that it is almost impossible to optimise one facet of the VE without detrimentally affecting another. It is therefore the aim of the designed DVE architecture to balance these requirements so that they are all realistically met.

1.4 VE Architectures and Components

Unfortunately, the technology associated with the design of VEs is still relatively new and not often freely available. Researchers tend to prefer to sell their technology to recoup loses incurred in setting up their own research and testing environments, while the companies involved prefer to sell complete solutions and are not willing to share their designs in order to be able to maintain their competitive edge over the other corporations in the field. The architecture designed, must for this reason, be created from scratch.

Basic VE architectures can be implemented using either the *Client-Server* or *the Peer-to-Peer* methodologies. In both cases, the architectures have disadvantages[4] that cannot easily be solved without using hybrid architectures. The hybrid architecture must therefore be appropriately researched and designed.

[4] The advantages and disadvantages of the Peer-to-Peer and Client-Server architectures are discussed further in Appendix B.

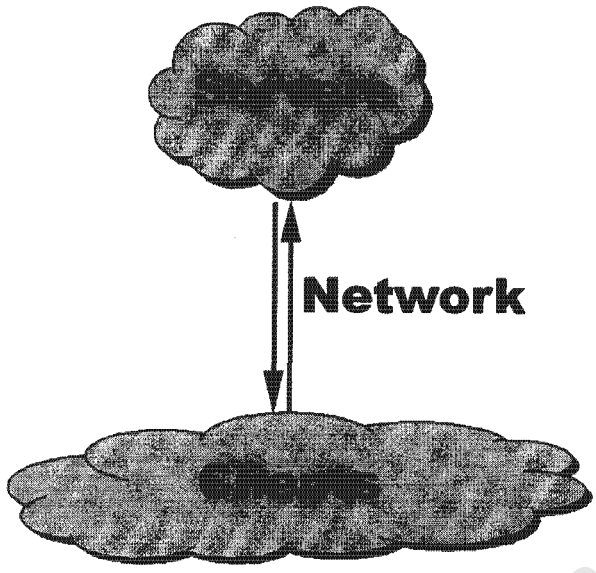


Figure 1.1: An abstract VE architecture

The hybrid architecture will consist of several components as simplified and illustrated in Figure 1.1. The *network* transports data between the *clients* and any *controlling servers* and must be able to supply the services needed by both the server and the clients. The server has several tasks ranging from brokerage to the broadcasting of information to the clients in the VE. Both the client and server terminals must be connected to the network and need appropriate displays, graphical or text based, as befitting the needs of the environment, and the processing power to process the data within the time constraints necessary to ensure the synchronisation and realism of the environment. The user terminals also need appropriate interface, control and communication devices to allow the users to communicate, move within and manipulate the environment.

1.4.1 The Clients

The interface with the user is an essential component of a VE and the various devices that can be used are of much interest[5]. Movement in, and interaction with, 3D environments seems relatively simple to us but our minds are capable of assimilating huge quantities of data from our senses, sorting and prioritising this data, and constructing a detailed overview of our surroundings. The aims of the various interface devices are to aid the VE in simulating the information that the senses require, and to aid in the creation of a realistic environment in which the user can believe.

[5] Not all our senses are as easily simulated within a VE and while we use all our senses to create a picture of what is around us, the senses of sight and hearing tend to be the most focused on. Appendix D focuses on various devices that can help users interact with the environment and with other users.

VEs require a lot of processor time with the processor spending much of its time accepting data from the network, calculating the changes to the user's view of the environment, updating this view and coordinating I/O to and from the user. If the user moves or wants to interact with the environment, the processor must also calculate the changes and notify the servers of the changes via their network connections.

The client software must be designed to ensure that it is capable of the timely completion of all these tasks to ensure that the realism of the environment is maintained.

1.4.2 The Servers

The servers in the VE are a key element in the VE architecture. The servers must be able to communicate with the clients such that they are able to receive data about the user's actions and transmit appropriate data about the other users. The servers must also be able to negotiate conflict situations within the environment where control is concerned. This all needs to be done within a realistic time-period as time delays within the server need to be kept to a minimum.

Further additions to the servers allow them to keep track of statistics that could be used for billing, server balancing and prediction.

1.4.3 The Network

The network in a VE must reliably transport the synchronisation and visualisation data that the users and any controlling servers generate, and do this in a 'realistic' amount time. Until recently, this support has not been possible on many systems due to the limited capacity of local networks that were also unable to handle many concurrent connections. For the same reasons the Internet is currently not a feasible networking solution and has resulted in DVEs tending to be limited to private high-speed networks.

The latest technological advances have heralded "High-Speed" networks that have allowed VEs the capacity to support many more users and have reduced the latency when delivering the larger quantities of data needed. Access technologies such as Asymmetric Digital Subscriber Line (ADSL), Hybrid Fibre Coax (HFC) and Fibre-to-the-Curb (FTTC), have been improved and allow users to

access many new services from home. This has made such tools as distance learning, conferencing and gaming, and their VR counterparts, that much more attractive to computer users around the world.

While not all Internet users have access to such high-speed, high-bandwidth technologies, the designed DVE architecture uses ATM natively. The advantages that ATM offers, when considering the design of a DVE, aid the design in ensuring the consistency, scalability and responsiveness of the environment and these advantages far outweigh the limited available of this technology. The rapid deployment, of these access technologies, is ensuring that the infrastructure is becoming far more readily available and this too reduces this disadvantage. Similarly, the demand for tools and applications that use the services offered by the new technologies is steadily growing.

ATM is a networking technology designed to transport multi-media information at high speeds through public and private networks in a cost effective manner. It uses fixed 53-byte cells to transmit data, allowing switch-based network architectures to transmit the information in multiples[6] of the OC-1 rate (51.84 Mbps). All computers connect to a switch, with communication between pairs of computers (connected to the same switch) taking place via dedicated connections through the switch. This is in contrast to such technologies as Ethernet that broadcast over shared media.

ATM is inherently **connection-oriented**. This allows connections to stipulate end-to-end **Quality of Service** (QoS) requirements and the connections inherently preserve cell sequences. These two attributes are particularly significant as the sending of environment data is often temporal and the sequencing of the data is essential. The data must also be delivered in a timely manner as delays degrade the usefulness and performance of the environment and the use of certain QoS specifications enable the meeting of these requirements.

As noted earlier, DVEs are dependant on the network to transport the data as rapidly as possible, and with 100ms or the user may become disorientated. Without the QoS guarantees, the environment cannot rely on the network being able to perform this data delivery satisfactorily. The QoS guarantees are negotiated at link creation and are guaranteed for the entire duration of the connection. This allows the network to monitor network usage and to limit or prevent congestion in the network that could degrade the services offered.

[6] Popular data transmission rates include: OC-3 (155.52 Mbits/sec) and OC-12 (622.08 Mbits/sec).

The transfer of data by an ATM network tends to be reliable, in part because ATM networks often user fibre optic connections that have very low error rates, and partially because of the error-correcting functionality built into the ATM cell *headers*. This checks and corrects any errors in the data headers but not in the data being sent. The further use of AAL5 as the connection's adaptation layer will ensure that the *data* being sent has a similar error checking and correction facility that enables received data to be checked for errors. This extra reliability does incur extra overheads although these overheads are balanced by the guarantees offered.

By **using ATM natively**, there is no need to use intermediate translation software to convert communication to ATM before sending. This reduces the overheads associated with the data being sent into the network. As the overheads can become rather large, the use of user defined AAL0 adaptation layer can also be considered. This is particularly useful for data that does not need the functionality offered by the various AAL layers. A VE design can profit from using AAL0 connections as the update data in a VE can often fit in 48bytes of space. Only single AAL0 cells are therefore needed to transmit data instead of several and this reduces the bandwidth and processing overheads associated with a single 'update' action for the environment. It is noted that updates may need to be sent many times a second and any small saving in this area will save a significant amount of bandwidth when considering the use of the environment over a period of time.

Multicast connections allow a single source to send to multiple destinations without the duplication of the data along the path of the data. This reduces the bandwidth consumption and reduces the impact on the network when dealing with a large number of diverse recipients. While multicast connections may seem useful at first, these connections only have limited use in the proposed design. This is due to the simple fact that a multicast connection is in fact a limited broadcast to a *set group* of destinations. In a DVE where the groups of receiving users can very rarely be segmented into *static* destination groups, the use of multicast connections is clearly limited.

Further information on the advantages and disadvantages of ATM can be found in [Min97], [Tat98], [Onv95] and an overview of various networking technologies available, can be found in Appendix C.

1.5 Dissertation Objectives and Development Methodologies

Most DVEs are commercial and thus not readily accessible to researchers. Further, the available DVEs are archaic and not suited to research within the domain of ATM networks. The aim of this

dissertation is to provide a development DVE platform on which further research can be done as to the usefulness of ATM networks in supporting DVEs and into the traffic characteristics of the data generated by such a DVE.

This dissertation presents the design of a scalable real-time interactive distributed virtual environment specifically targeted for use over an ATM network. The designed architecture offers a reliable and believable service to the user by:

- Setting out QoS guarantees that the network and server applications must maintain to ensure that the DVE remains responsive to the user's requests.

- Supporting a distributed server architecture that allows for scalable growth of the VE. The distributed service is hidden from the user by broker environment connections to simplify the user's interaction with the environment.

- Ensuring the reliability and consistency of the VE through duplication of the VE data across the servers in the distributed architecture.

- Using management algorithms to reduce the quantity of data that must be sent or received within the environment, and by efficiently using the network resources available.

- Providing a means for the location of the distributed servers through an abstract interface as well as a means of tracking the users across the servers to allow for security, billing and other client statistics.

The architecture has been named HEAVEn and is a self-referencing acronym for *HEAVEn is an Experimental ATM-based Virtual Environment*.

The initial phase in defining the problem of creating a VE, or DVE, covered the investigation into the end-to-end needs of the client who would connect to the environment. Research revealed that the clients have very stringent delay requirements as well as reliability concerns when transferring data across the network. In addition, the clients using the VE can generate large quantities of data and steps to reduce the quantity of this data where needed. These steps could also not detrimentally effect the

end-to-end transfer time or increase the error rate. The quantity of data and the quality of the network services required make legacy networking technologies almost useless and it affirms the need of a QoS aware networking technology such as ATM.

The next phase involved deciding how the clients would be interconnected. In its simplest form, there are only two ways to interconnect clients: Peer-to-peer, where each client connects to the other clients, and client-server, where the clients are connected to a centralised server through which all communications pass. While a peer-to-peer architecture seems to be a very useful scheme, the reliability and consistency of such a system is too unpredictable due to the temporal nature of the peer connections to the environment. Without reliability, the environment will not be able to remain consistent between all the users and a persistent environment cannot be ensured. A hybrid of the two schemes was therefore chosen to be the most effective solution. The client interaction with the environment is in a client-server manner and the interaction and distribution of the centralised *Environment Servers* is peer-to-peer. The servers are fully interconnected (a fully connected mesh) to ensure that communication between peers servers can take place directly and no single server is reliant on any other servers and this increases the reliability of the design.

To ensure that the complexity of the peer-to-peer server mesh is hidden from the users and the client-server approach maintained, a distributed fully connected mesh of *Broker Servers* is used to abstract the 'centralised' environment servers and transparently direct a client to the appropriate server. The broker that is directing the client, being in contact with the server mesh, is able to direct clients to servers in an intelligent manner and balance the load on the servers. Further, the broker is able to bill clients, collect statistics and store a database of customers, security codes and clearances that are used to access the VE that the broker is serving. A *Name Server* is used to direct the client to the appropriate Broker Servers for a specific VE and abstracts the ATM address information.

The environment servers, while abstracted as a single server, are not simply meshed, but consist of several servers each offering specific services to the client. The *Broadcast Servers* accept update information and distribute this information to the clients that need the information as well as the other connected broadcast servers who in turn, distribute the data to their clients. The *Control Servers* accept requests for control of objects and negotiate this process. They also control the creation and destruction of objects, announce disconnecting and connecting clients and govern any other controlled interaction that needs to take place within the environment. The *Model and Media Servers* store and distribute object data to clients. The data describing objects within the environment (a video clip, for

example, could be several megabytes in size) can be large and to ensure that the users do not need to store *all* of this data locally, there needs to be a data retrieval service.

The design defines the connections between servers, and between servers and clients. As connections have differing requirements, the services offered by the connections must match the needs of the data being transported. This must be considered to ensure that the transport of the data does not adversely affect the environment. Similarly, the end-to-end requirements for the communications are also considered. This ensures that latency, specific to each element in the communication path, has an upper bound and a maximum jitter that can be guaranteed. This ensures that communication between clients remains immersive.

The next phase focuses on the implementation of the design in code. The implementation was completed on the *Windows NT 4.0: Service Pack 6a* and *Windows 2000: Service Pack 3* operating system platforms using Visual Studio's Visual C++. Winsock2 [Sta96] [Sta97] was used as the communication interface to the ATM network.

With the design implemented, test results were collected. As the design ensures that the environment databases remain consistent, specific results relating the scalability of the environment to the latency of the connections were measured.

In the final phase, conclusions as to the usefulness of the design and the implementation are made and possible extensions and further work is presented.

1.6 Scope and Limitations

The HEAVEn project is a combination of efforts by Kevin Dennis and Stuart Doyle to design and implement an architecture using the specific services offered by an ATM network. The architecture can function using another network protocol for data transfer although certain inherent attributed and services found in ATM would need to be emulated in software, thus increasing processor overheads, if they were not already available in the new system. While this is true, the specific guarantees offered by an ATM network, in terms of response time guarantees, reliability, data ordering and congestion management, are particularly useful in ensuring the consistency of the environment and the immersive experience of the environment for the users.

The foci of this dissertation are the server and network architectures needed to support and meet the requirements necessary for a real-time interactive, consistent, scalable VE. While scalability is a major factor in the design, due to limited hardware resources the testing of the scalability of the design had to be done using multiple simulated clients. The implementation covers the basic control and broadcast components as well as the brokering, name resolution, and model and media distribution services. The creation of software libraries that offered simple and efficient access to the ATM network was an initial priority.

The related work by Stuart Doyle considers the design, implementation, usability and interaction techniques necessary for the client and client-side rendering of the environment. It also focuses on methods to reduce the quantity of data that must be transmitted by both the clients and the servers. This includes the group management functionality and the use of client-side prediction algorithms. As the clients interact heavily with the broadcast and control components of the architecture, the implementation of these components was a combined effort.

For simplicity, it is assumed that the simulated client software and hardware is capable of accepting and using the data being distributed and that the clients only transmit useful data. The limiting of the data sent and received by a client and the negotiation of these needs with the various environment servers is covered, as noted earlier, by Stuart Doyle. To ensure that the latency requirement for the environment can be set out, it must be assumed that a client is capable of accepting sent data from the environment servers, process this data and update the user's view of the environment within 10ms. Similarly, the client must be able receive and process user input requests, and initiate data sends to the environment servers, within 10ms. These assumptions are based on related work [Wlo95] although the rendering engine used and hardware interfaces are capable of meeting these limits if the client data processing is sufficiently optimised The client is also assumed to police its own output traffic and to obey the QoS contract negotiated for its connections. While this need not be the case, it is in the client's best interest to do this as traffic that does not obey the contract could be discarded by the network and thus prevent the client from successfully interacting with the environment.

The Operating System (OS) used for the development of the DVE architecture was *Microsoft's Windows NT* and *Windows 2000*. While neither of these are ideal RT platforms, they were chosen because the client-side rendering engine code and licensing agreement were only available for those operating systems. The choice of the programming language was simple as WTK's Applications Programming Interface (API) functioned best when using Object Oriented (OO) *C++* [Bud02]

[Min97] and *Microsoft's Visual Studio* was available as the programming environment. The use of an OO language when designing software is important to the implementation modular design. This ensures that the software is able to be easily developed and enables simple additions and alterations to the existing systems.

1.7 Contributions to the Field of VE Development

While there are a large number of virtual environment architectures that have been and are currently being developed, implemented and researched, none of these designs chose to use the native advantages offered by an ATM network. The HEAVEn architecture uses these advantages to create a scalable distributed real-time interactive virtual environment that offers an immersive experience to the user and ensures a consistent environment.

The HEAVEn architecture proposes the separation of the Control, Broadcast and Model distribution functionality of the virtual centralised environment server and the functionality of each of these servers is discussed thoroughly.

The HEAVEn architecture includes the use of *descriptor files* to hierarchically define and divide the environment to enable a simple method of defining levels of detail that the client can display, and to limit the size of the portions of the database that the client must retrieve. The descriptor files also direct clients to other descriptor files that represent adjacent, contained or encompassing areas that the client software can choose to retrieve and cache.

Certain other concepts were original ideas although later research located architectures that have similar conceptual components. These include the use of the broker servers to balance the load and to collect and collate data about the user's actions within the environment.

1.8 Document Structure

The thesis will be segmented as follows:

- Chapter 2 focuses on work that has been done in the field of VEs. The aim of this chapter is to point out aspects of the various architectures and note where they have failed or excelled. These aspects will be then be focused on in Chapter 3.

- Chapter 3 explores the various functional design requirements of a VE. While the introduction covered the five main features of a VE and several of the problems that are introduced when networking a VE, these attributes need to be further defined in terms of the functionality that the actual design needs, before the HEAVEn architecture can be designed. This chapter will present a fundamental set of needs that any designed VE incorporating network and server applications, must be able to maintain and offer to the users.

- Chapter 4 presents the design of the HEAVEn DVE architecture and its components. Reasoning for specific design choices are given here, with references to the criteria described in Chapter 3. While the design is modular, the design must perform well as a whole and the modular descriptions note where design trade-offs have been made. Issues focused on include the methods used by the design elements to discover each other, the justification for using a meshed topology to interconnect portions of the design, the means of limiting and balancing congestion and load at the servers, the implementation of customer billing and statistics and how information security is achieved.

- Chapter 5 focuses on the implementation of the proposed DVE architecture and the results achieved through the implementation. While the HEAVEn architecture, described in Chapter 4, specifies how the DVE is to be constructed, the implementation details are specifically left open-ended to ensure that the design can be implemented across multiple platforms. This chapter notes implementation choices made, problems encountered and their solutions, enforced modification due to implementation problems and modification to the design made to simplify the implementation or improve the performance of the design.

- Chapter 6 presents the conclusions drawn from the implementation and design and the recommendations for future work. The dissertation concludes that, while the HEAVEn design is able to support the response time requirements of a DVE and, using ATM based technologies, offers several advantages that further ensure the reliability of the system. Further work must be done to limit the quantity of data that is distributed within the environment as without this limiting, the virtual centralised server will become congested and the service offered to the clients, compromised. The recommendations focus both on work to find

solutions to solve this problem in the work that is still being done on this project, and on work to add to the functionality and usability of the DVE architecture.

- Appendices are included to provide background information covering networking architectures, the current networking technologies available and VR hardware and software.

Chapter 2 **Related Work**

While VEs only became popular in the 1990s, the concept has been around for some time and researchers began working on the problem of creating realistic computer simulations, mostly for military training, as early as the 1970s. These designs, used to train people in a "safe" environment, soon became the source of much competition. As networking technologies advanced, so did the capabilities of the VEs, with the American military being the first to use these new capabilities.

This chapter discusses a number of VE architectures that have been designed and implemented, focusing on the various known advantages and disadvantages. The aim is to isolate problem areas and focal points that will need to be dealt with in the proposed HEAVEn architecture.

2.1 SIMNET

The first successful implementation of a large-scale RT networked VE was the Simulator Networking (SIMNET) project [Mil95]. A group consisting of both American military and commercial researchers developed SIMNET and the Distributed Interactive Simulation (DIS) standard that developed from it for use by the military in training simulations. This work started in 1983 with the final delivery in 1990.

While SIMNET was an outstanding achievement at the time, there were several problems. The architecture was limited in its use as it was designed as a military training simulator and this prevented SIMNET being used in many other areas. The interface only allowed users either to control large-scale troop movements or to control a military vehicle, normally a tank. The architecture also limited the number of simultaneous users in the environment, to between 50 and 200 [Ste93]. While this was an achievement at the time, is still a significant drawback when.

The architecture made extensive use of broadcasting, and later multicasting, techniques available on early Ethernet networks, which used a shared transport media, to distribute data in the peer-to-peer architecture. Using the broadcast mechanisms enabled the architecture to send data to all the entities in the environment although this forced each entity to process and filter the data received even if it did not need it. This extra processing time was a problem as it limited the client entities although this was considered a worthwhile trade-off at the time. Further attempts to increase the size of the LAN used

using routers and bridges only resulted in increased delays and even more congestion within the network.

While using broadcast techniques was an advantage at the time, these techniques became redundant when the advances in switching technologies and independent media allowed architectures to send data to specific entities within the environment, thereby optimising network traffic. This could not be taken advantage of because of the broadcast dependence of the SIMNET architecture.

Another interesting aspect of the design was that SIMNET used a *Dead-Reckoning* scheme to reduce the quantity of data that each client needed to broadcast onto the network. The data sent in this scheme contains the current position, orientation, velocity and acceleration of an object within the environment. As long as the object continues moving within the path as predicted by the algorithm used by the Dead-Reckoning scheme, no other information needs to be sent. Since the source of the information used the same prediction algorithms as the other entities within the environment, it calculated where the object should be and if this differed significantly from the place where the object was, a new packet of data was sent out to the other entities to update the object's position, orientation and movement. This could result in large discrepancies within the environment database stored locally at the clients if the update information about critical events was lost. The trade-off was that the scheme significantly reduced the bandwidth needed.

This resulted in the data being far more critical to the environment's believability as a lost packet would have a far greater effect. The only way to ensure that data would not be lost was to prevent congestion within the network by limiting the number of users and by using a dedicated network.

It is important to note that the SIMNET architecture's faults lay in the fact that it did not limit the data sent to its clients on a per-client basis, choosing instead to send all the update data to every client. This resulted the client software becoming the bottleneck. It also resulted in a 'weakest link' problem where the weakest client became a bottleneck and there was no way of increasing the overall performance of the environment without updating all the clients. Similarly, the network became heavily congested due to the broadcast means of transmitting data and this is clearly not ideal in a non-dedicated environment.

2.2 DIS

Based on the success of SIMNET and due to its operation-specific nature, which limited its usability, the creators continued their research and developed DIS as a far more general-purpose system. DIS's communication protocols are defined in the IEEE 1278 DIS standard [IEE93].

The DIS architecture expanded on the user's capabilities by allowing a user to take part in the simulation as a soldier as well as a vehicle operator or troop commander. The design was also modified to allow a greater number of simultaneous users, approximately 300 to 500 entities, and to make the interaction with the user far more independent of the user's interface software or hardware. Several architectures have used the IEEE DIS Protocol Data Unit (PDU) format, described in IEEE 1278, as a basis for their communications. The generic nature of these PDUs enabled not only user-controlled participants to interact in the VE, but software controlled clients and even real machinery. Unfortunately, this same generic nature makes the PDU rather large and wasteful of bandwidth.

The DIS architecture still used Ethernet's broadcast technology that limited SIMNET. Where multicast capabilities were available, a single multicast group was constructed to include all users and servers. While this isolated the environment data and limited the design's impact on the network, it did not improve the congestion problems associated with the design's scalability. As early multi-user simulation architectures, both SIMNET and DIS operated well and effectively, but as the need for larger simulations and for more up-to-date technologies grew, they quickly became limited. Their main problem was that they made extensive use of broadcast communication, and as such were restricted to running on broadcast capable LANs.

Another important aspect about the design is that it relied on the user to update information about its actions. State changes needed to be broadcast to all users in the environment and the clients needed to ensure that a *heartbeat* update was regularly sent to ensure that the models in the environment remained active. Objects without a heartbeat were removed from the environment. While this might seem a practical method, consider a user who has been 'shot'. If the user does not broadcast that it is wounded, then no one will know this and will assume nothing has happened. Simplified, this is a lack of security in the environment as there is no centralised authority governing critical interactions. Similar situations arise when objects within the environment are disabled but due to data loss or clients that are ignoring certain information, the objects continue functioning with no overall controller

preventing this. While certain of the situations can be resolved when the object heartbeat next updates everyone, assuming it does this correctly, the security issue is still a problem with the DIS architecture.

The large quantity of data in the DIS PDUs was another drawback as it very easily cluttered the network causing congestion and inconsistencies within the environment. Where the PDUs were originally constructed for accuracy and with data duplication in mind, this extra data was particularly wasteful of network bandwidth. Several papers have been written regarding the simplification of the DIS packet structure [Coh94a] [Coh94b] to as much as 20% of its original size.

It is important to note that any future design should endeavour to ensure that data about objects within the environment is only updated when it is necessary as obvious duplication of the data can have serious congestion effects on the network. However, if duplication is used, the redundancy of the information can be used as a means to increase the reliability of the environment as small data losses can be tolerated using simple client side smoothing algorithms.

Data updates must also be limited to needed information to prevent the transmission of extraneous data that simply wastes bandwidth. Minimising the amount of data sent can significantly affect the overall scalability of the environment. When using ATM technologies, this translates into an implementation that would fit all needed update information into a single transmit 48-byte cell (assuming AAL0 communication is used).

2.3 NPSNET

NPSNET consists of several generations of software designed and implemented by students at the Naval Postgraduate School [NPS02] and is aimed at human-computer interaction in large-scale VEs. While these predecessors were a miss-match of component systems put together by the students, the first NPSNET was a highly funded project run in conjunction with SIMNET with the aim of creating a better DIS-compliant (using the IEEE 1278 DIS PDUs) system. NPSNET's development continued in parallel to the SIMNET and DIS projects and became the first VE to use the Internet [Mac94] [Mac95] as its transport media using a multicast backbone to multicast data between its clients.

Unfortunately, being based on the DIS standards meant that NPSNET suffered from many of the PDU related problems detailed earlier although research was being conducted into the 'slimming' [Mac94] of the DIS PDUs so that they could be sent over an ATM network - unfortunately, no results of this

research have been found. NPSNET did offer better bandwidth management due to the use of 'multicast groups' to group users into destination sets. Each group was able to receive group specific data which limited the quantity data sent to each client. While the design itself was not truly scalable, the grouping concepts have been extended in many VE designs and will need to be seriously considered in the HEAVEn design. NPSNET V is currently being developed. The new design allows the software to be multiplatform and enables the architecture to be tested over a much broader range of hardware.

This architecture is designed for use on multiple platforms. This interoperability is essential to a scalable VE design and it is important to make allowances for users who are using systems with minimal specifications and users that have high-end systems.

2.4 Paradise

While most academic projects focused on the graphical interfaces to VEs, the Performance Architecture for Advanced Distributed Interactive Simulation Environments (PARADISE) [Hol02] project focused on network issues with the aim of reducing the bandwidth used. PARADISE used multicast connections between the clients with one multicast address for each object in the environment and a set of *Area of Interest* (AoI) Servers that directed the membership of the multicast groups.

PARADISE also focused on methods to improve the dead-reckoning algorithms used in predicting object motion as limiting the need to send data, further reduces the bandwidth consumption. The architecture also supported data transmission at varying rates to ensure that slower objects need not send updates as regularly as rapidly moving objects would need to.

Using multicast connections was still not reliable and to limit the need for a heartbeat for each object to ensure that lost data is recovered, a reliable multicast layer was introduced to ensure data delivery. Users joining multicast groups would have to request the current state of the environment from a system of *Logging Servers*, which captured the state of the environment at all times.

The design changed many of the common assumptions about VEs that DIS, SIMNET and NPSNET had made and offered the new concepts of using the AoI to manage the multicast connections while assuming the multicast connections to be reliable and thus not needing heartbeat update information

that could at any time be retrieved from the Logging Servers. Unfortunately, the design was limited by the graphical capabilities to between 50 and 70 entities.

It is important to note here that the design was able to limit the data sent to a specific user and by doing this, allow each user to specify their needs and the area around them in which they are interested. The inclusion of AoI limiting of data will be an essential tool to limiting the data and in specifying each users QoS needs when using an ATM connection between the clients and the servers in the HEAVEn design. The inclusion of the reliable multicasting layer, while important to the usability of the design, was a further layer through which data had to travel before being accessible to the users. This reduced data throughput and was one of the limiting factors when considering the scalability of the design. Limiting the number of stages that data must go through before it is usable to the users is clearly a factor to consider when designing a DVE architecture.

2.5 DIVE

The Swedish Institute of Computer Science created the Distributed Interactive Virtual Environment (DIVE) [DIV02], designed as a VE in which users could interact and collaborate on certain tasks. The design philosophy was that such an environment would typically have fewer human entities than other objects and that this would result in entities having to manage the distribution of large quantities of data and this management would have to be streamlined.

DIVE breaks its 3D space into "worlds," each a separate and isolated group within the VE and all data about a specific world is duplicated to every entity in the world in order to facilitate the collaboration. While this means that the data must be replicated to all those within the "world," this is simplified by using IP multicasting.

Since full replication of the data is required to ensure the concurrency of the environment, the updates must be reliably transmitted. Early versions of DIVE made use of a reliable multicast protocol based on positive acknowledgements and due to the overheads that this created, even the later versions were able to support only between 16 and 32 users at 200ms latencies [Mat97b]. The replication of the environment data across either the servers or the users is a design choice that will need to be further developed. The division of the environment into 'worlds,' must also be considered. It might seem like an ideal solution to limit the data being transmitted in the environment, but when one considers that all it is doing is creating separate environments that themselves have the same limiting factors associated

with a single environment, it is clear that this is not the solution and other means of limiting data being sent in the environment will need to be considered.

2.6 BrickNet

BrickNet [Sin94], unlike the previous architectures, divides the VE into interconnecting worlds. Each world is controlled by a server with all communications between the clients in the same world being mediated by the central server. The centralised server, while useful in the mediation and security of the VE, causes latency and processing bottlenecks that limited the architecture to between 8 and 32 users. The client/server approach however offered the environment much higher security and database consistency while limiting the quantity of data that the clients needed to store. This is the first of the architectures that does not distribute the entire VE database to all the clients. This is important, as the environment must be scalable as it could become large. It is clearly also a trade-off due to the choice of a secure limited distribution of the database versus the bottlenecks the centralised server creates. The limitation here is that the environment is simply a collection of single server systems that have bottle-necks and single points of failure at the central world servers. This concept would be a serious design flaw if used for a DVE that must be consistent and persistent.

2.7 HLA

The High Level Architecture (HLA) [Def02] extended the work done by SIMNET and DIS and took into account many of the newer networking, hardware and software technologies. The main difference between HLA and its predecessors is that HLA specifies only the interfaces necessary for the components in an HLA VE and not the implementation details. These specifications could therefore be implemented over an ATM network although the specifications themselves do not take into account the unique features of ATM as they were designed with an IP architecture in mind.

Some of the functionality defined in HLA covers the joining and leaving of the VE, forwarding of data between entities within the VE, grouping of entities within the environment and the adding and removing of entities from these groups. Grouping of data is clearly important to limiting the unnecessary duplication of data within a networked VE and the options offered by the HLA could be worthwhile additions to the design.

2.8 MASSIVE

The University of Nottingham created the Model, Architecture and System for Spatial Interaction in Virtual Environments (MASSIVE) with the aim of creating an architecture to support local interaction [Gre95]. It does this by specifying an *aura* for each object dependant on the type of interaction that is taking place (examples being graphical, textual, sound). The aura specifies the logical area around the user's avatar that these interactions will affect. If an aura, belonging to one user, reaches the interest area of another user, then an *aura manager process* will notify these two peers and point-to-point communication will be set up between them to transfer the interaction data.

To measure the interest areas of users, the architecture specifies a user's *focus* and *nimbus*. The focus is the area that the user is focusing on and the nimbus is the area around the user in which they are interested. These two characteristics allow dynamic AoI calculation and QoS specifications specific to the user.

Connections use standard IP point-to-point allowing multiple data streams to be sent over one connection between peers. No multicast support is offered, so large groups of peers in a single area could cause congestion, especially in the case where one user is transmitting to multiple recipients. Since the client is not geared towards this distribution of data, congestion could easily occur. A similar cause of congestion could be the aura manager that has to receive and process position updates from all users within the environment. It needs to do this to allow it to calculate the various aura collisions and while the updates can be collated and filtered, they will still cause congestion as the aura manager is a single server in a loosely client-server architecture. Due to the use of IP and the congestion caused at the aura manager, the system was not scalable and could only support at most 10 simultaneous users.

2.9 mWorld

mWorld [Dia97] is a TCP/IP based VE architecture using the Joint Editing Service Platform (JESP) developed through several European Union projects. It uses Silicon Graphic Incorporated's *OpenInventor* Objects and VRML to provide 3D editing, animation and navigation. The platform is a fully distributed peer-to-peer architecture that guarantees data delivery through the JESP interface that in turn implemented its TCP/IP stack over an ATM network.

The modular approach to the design that is enhanced by JESP ensures that additions to the architecture are transparent to the users and allow simple modification. Unfortunately, due to the reliance on the TPC/IP protocol, the design is still not able to offer per connection QoS although the use of ATM as the networking layer can ensure that the data sent between peers is delivered in guaranteed time.

Unfortunately, this design did not consider the use of native ATM to transport the data as this would have been analogous to the use of UDP in the IP domain thus reducing the overheads associated with data transmission. The use of native ATM as the transport media is one of the strengths of the HEAVEn design.

The use of JESP as an intermediate data transport layer increases the latency in the design, as the data being transferred must travel through the layer with its additions and checks before being sent and received. While this does guarantee the data delivery, the introduced end-to-end latencies limit the environment as less time can be given to the processing of environment data forcing the clients and servers to be more powerful if they are to remain capable of supplying a realistic RT experience to the users.

2.10 VENUS

VENUS (VE Network Using Satellites) [Uda98] is a hybrid architecture that uses both peer-to-peer connections between the clients and servers to distribute data. The data is seen as being divided into two components that represent *general* information and *specific* information about the user's environment. General information about the user's actions is sent to a *broadcast server* where it is then processed and broadcast to the users connected to the environment. The specific data, which would be of interest to only a few users, is sent via direct peer-to-peer connections.

While this hybrid design does offer significant advantages over other VE architectures, it assumes that a user will only be needing specific information about a few users and that the latency between the users is short enough to maintain a consistent view of the environment. Both these situations cannot be assumed in a general purpose DVE.

2.11 World2World Release 1

Sense8's "World2World is a client-server based networking solution for 3D interactive simulations" [Sen97, Pg 5]. The system design revolves around the use of a client-server design to which clients

connect, to ensure synchronisation and simplicity in deploying the system over Wide Area Networks (WANs) and the internet.

To ensure the limiting of data to the users, all data passes through the centralised server where the data is time-stamped and queued with only the most modern information being sent to the user at a user specified interval dependant on the user's network connection, processing power and graphical rendering ability. This could result in congestion at the centralised nodes. Multiple 'simulation servers' are possible in the design although the specifications are limited as to their interconnectivity and this distribution of the simulation server could alleviate problems associated with the necessary processing of the data packets.

The World2World overview architecture is similar to that of the HEAVEn architecture in that both are designed for use in a wide area environment. To ensure database consistency, a client-server approach is used with the option of distributing the centralised server. Data is also collated in a similar way to allow the centralised server to maintain the user's QoS requirements.

The design also provides for 'server managers' that broker new connections to the environment and this prevents the environment from being delayed when new clients join the environment. The client initially connects to the broker before being handed off to the main environment server.

World2World uses its own RT protocol that has been implemented on top of the UDP protocol to avoid the overheads associated with the TCP protocol. The bespoke protocol bundles acknowledgements with data to be sent and resends lost data only if modern data is not available to be sent. While the design functions well, the reliance on the UDP protocol prevents the architecture from guaranteeing service to the clients beyond the best effort service supplied by the UDP protocol.

2.12 Caterpillar's Distributed Virtual Reality

This project is of interest as it is used by many VE critiques as being a VE design that uses an ATM network to transport its data. While information is not readily available regarding this project due to its commercial nature, it is known that the design does use an ATM network to transport data. It does this by using various unicast and multicast IP connections over the ATM technology [Cat98] and this severely limits the advantages that ATM offers.

2.13 Summary

There are many VE architectures available and the above examples are a sample of these designs. The descriptions above aim at highlighting the various weaknesses and strengths so that they can be collated in a set of functional requirements that need to be focused on in the HEAVEn architectures.

While many other VE architectures are available and each deserve recognition, research into these architectures shows that they all feature similar weaknesses and strengths and are based on similar technologies to those focused on in this chapter. Some of the other architectures that have been researched are SPLINE (and Open Community) [And95], AVIARY [West92] [Sno94] [Sno96], WAVES [Kaz93], RING [Fun95], DOOViE [Ahn97], Urbi-et-Orbi [Fab00], Community Place (also known as Sony's Virtual Society) [Lea96a,96b,97] [Hon96], Parsec [Var98], COVEN [Cov96], CoRgi [Ror99] [Mun99], NOMAD [Wil01], Maverick and DEVA [Pet00] and several others. Where possible, papers relating to these architectures have been included in the Bibliography as further reading.

Chapter 3 **Functional VE Requirements**

"In other words, [VEs are] about fooling your senses into thinking that you are experiencing something you are not, in an environment that doesn't exists, while interacting with people, animals and objects that [also] do not exist." - Bianca Wright

The design of a scalable interactive VE has many requirements that cover several areas of expertise. In a design that incorporates so many disciplines, an approach that improves the performance in a specific area, may well severely detriment more than one other area. This is often the case in the design of large-scale systems where performance is an issue in more than one area of the design. The VE architectures discussed in Chapter 2 vary considerably and focus on many of the key factors that need to be taken into account when designing a large-scale RT VE. In order to get a working, useable system, trade-offs need to be made during design. These trade-offs are not intended to reduce the performance or requirements of the system, but rather to provide the right balance of functionality that will lead to a solution that meets the requirements in all areas.

While the five main features offered to the users in a VE are Space, Presence, Time, Communication and Manipulation, as discussed in the Introduction, these features need to be translated into specific focal areas. These areas can then be considered in the design process and that translate well into services that the DVE can offer and included in the designed architecture in Chapter 4.

3.1 Scalability

When considering the design of a large-scale networked VE, it is important to understand that the concept of scalability is multifaceted. Scalability can be considered in terms of the number of possible connected clients, the bandwidth used for communication and the ability of the environment to offer its services to clients with varying hardware and software capabilities. Clearly, scalability covers a wide range of options but it is important to realise that the source of the need for scalability for VE's is the need to support large numbers of simultaneously connected users. The other scalability requirements follow from this need.

The main concern for the design is to ensure that the environment remains responsive despite any increase in the number of connected clients. This will obviously included bandwidth and processing

scalability that will allow the various environment components to reliably support the increased number of clients wishing to connect to the environment.

3.2 Environment Consistency

Consistency within the VE ensures that all environment nodes, users and servers, within the VE share the same view of the VE. Inconsistencies arise when environment nodes have different views of the VE and the differing views are most often caused by loss of data during communications. The differences could result in users taking different actions with respect to the current situation and could result in conflicts between users. This can have severe repercussions when considering the realism and immersion of the environment.

Methods to ensure the consistency of the environment need to be included and if necessary. Regular checks can be made to the environment data or the transmitted data can be guaranteed by using an acknowledgement protocol to ensure data delivery. The problem of database inconsistencies at the various environment nodes is a problem peer-to-peer architectures suffer from, as there is no centralised control of the environment and no mediator to negotiate conflicts.

3.3 End-to-End Delays (Latency)

A standalone VE on a desktop computer has almost negligible delays between actions and responses. Conversely, the delays in a networked VE can be considerable as the clients are often geographically distributed. Research has shown that for time-critical communication within a VE, delays of longer than 100ms break the user's experienced immersion in the VE and can even make the user feel disorientated and sick (especially when using such tools as HMDs). The design needs to *ensure* and *monitor* the delay to prevent the delays from increasing beyond the 100ms barrier and take appropriate action to prevent this.

It is important to realise that the delay is specified as an *end-to-end* delay and the 100ms [Wlo95] [Str00] limit (or bound) is in fact the maximum delay between the action a user initiates and the response as seen by the user within the VE. This limit must include time taken by the client to render, process, receive and send data about the environment, the time taken for data to travel across the network and the time taken for the servers to send, receive and process any data.

The IEEE DIS specifications [IEE93] state that the end-to-end delay in a VE should be no less than 100ms and that there should be no more than 10ms jitter in the data received. This standard should be used in the design as a quality assurance guideline.

The latency and end-to-end delays are a factor of the networking technology used and therein lies the benefits of using ATM directly. The largest single source of latency in the network is the time spent traversing the protocol stacks required by the various communication technologies. By eliminating the need for the other communication protocol stacks, the processing time required, to send or receive information, is greatly reduced. Using ATM directly also improves bandwidth efficiency by removing the overhead associated with the extra header information added by these protocols. While a single protocol's packetisation mechanism might not necessarily affect the services provided by that protocol, the overall effect of using multiple protocols can be detrimental to the services provided to the application.

Using software layers to simplify environment design can be very useful in certain situations but in the design of a time-critical application such as a VE, where end-to-end delays must be minimised, these intermediate layers add latencies that cannot be tolerated. Such layers include the extra overheads introduced by TCP when the data could be being sent over a UDP connection, the overheads included in JESP as used in the mWorld Architecture and the overheads introduced by CORBA [Wil01] or other intermediate software layers. While CORBA, JESP, TCP and other 'middleware' additions do offer significant advantages to the development of the VE, the extra latencies make it difficult for the VE to be truly large-scale and the use of any intermediate layers will need to be carefully considered.

3.4 End-to-End Reliability and Fault Recovery

The reliability of all the components within the VE allows the environment to remain consistent. The clients, servers and transmission media can all cause errors in the environment and the design needs to be able to recover *gracefully* from these errors or prevent them from occurring in the first place.

3.4.1 Catastrophic Errors

Such a situation could be caused by the failure or loss of an environment server, the failure of portions of the networking infrastructure or the failure of the client software or hardware itself. The recovery from certain of these situations is possible if the client is still able to communicate and reach other appropriate servers. While it is possible, this class of error tends to cause delays that are longer than

the 100ms limit and the user will need to be notified that the environment is being *paused* while attempts at reconnection are made.

3.4.2 Other Errors

If the error is not catastrophic, then the design needs to be able to recover from the loss of the data, the errors in the transmission of the data (correctable or not), the arrival of older than current data and data that has been duplicated.

Tools to enable this recovery include *error-detection* and *error-correction* codes that ensure that the erroneous data can be detected and/or corrected. To ensure that data is not duplicated or applied when it is old, *order* and *time* information can be included. As data to be sent can be critical to the consistency of the environment, this data can request an *acknowledgement*. If no acknowledgement is received, the data can be re-sent. In addition to this, the clients can *scrub* their environment database by synchronising it with the peers or with an environment server.

It is also important to note that not all data is crucial to a client's view of the environment and needs to be recovered. Consider movement along a path where updates as to the position of the object within the environment are sent every half a second. Should one environment update be lost or damaged beyond recovery, there are several means of dealing with the problem:

- The *data loss can be ignored*, resulting in a slightly larger 'jump' in position as seen by other users, when the next update occurs. These jumps can be smoothed by using simple prediction algorithms or by smoothing the motion of the objects within the VE between updates.

- The *data loss can be recovered*, resulting in a delay as the data is resent as the application of updates must be applied in order. This delay could compromise the environment's end-to-end delay requirements.

Update information is not always critical to the continued functioning of the VE although there is critical data. Consider the data where a user stops or starts movement (*drastic change*), or where a user requests control of an object or releases it (*object control*), or when a user enters or leaves the environment (*environment alteration*). In these cases, loss of the information can result in large

discrepancies in what is perceived in the VE and the update data notifying clients of these changes must be error free and guaranteed.

3.5 Bandwidth Efficiency and Data Limiting

Network bandwidth usage must be minimised in any designed VE. Limiting data transfers by using compression techniques, prediction algorithms, level of detail and area of interest algorithms, and selective limiting of the data through user specified rate management, must be considered if the bandwidth usage is to be successfully limited. There are several reasons for limiting bandwidth:

- The cost of network bandwidth must be kept to a minimum if the design is to be commercially viable. Not all users are able to afford high bandwidth connections thus lower bandwidth connections will tend to be the standard in a public VE.

- Both the clients and any servers in the design will have their bandwidth limited by the network interface cards. While the bandwidth can be considerable, it will eventually be used up and a bottleneck will occur.

- While algorithms can be implemented to reduce quantity of data being sent, these algorithms will increase the processing overheads associated with the sending and receiving of the data thereby increasing the delays. These delays could affect the end-to-end limitations detrimentally. Tradeoffs will need to be made to accommodate both situations remembering that any client-side algorithms will force increased processor requirements on the client and limit the range of clients that can connect to the VE.

There are many methods to reduce the quantity of data being sent, examples of which are noted here:

- *Compression algorithms* can be used to reduce the size of the data being sent however, the use of these algorithms will cause overheads that could create a processing bottleneck especially in client situations where low-end processors are being used. Compression may be offered on a per user basis dependant on the requirements of the users involved and the process time available to both the sender and receiver.

- *Prediction algorithms* [Ber01b] have similar problems as they also introduce extra processing overheads. While prediction algorithms do enable less data to be sent, they also increase the receiver's dependency on the arrival of the data to ensure consistency. The prediction removes the redundancy of information being sent with the resultant effect that data errors affect the receiver's view of the environment far more seriously and endanger the consistency of the environment.

- *Area of Interest* (AoI) algorithms (also known as Field of Interest, Importance of Presence [Oh97], Field of View, Occlusion and several others) can be used to limit the logical area within the environment from which the user's client will receive information. The received data is filtered [Roe99] dependant on these needs and this is usually done at the peer-sender in a peer-to-peer environment, or at the centralised server in a client-server design. The calculation of the filtering rules is often computationally expensive. Clients should therefore not be forced to calculate these AoI lists as this could limit the user's experience of the environment if they were using a system with minimal processing power. The central server is ideal for this as it also has all the necessary information about user positions within the environment and can distribute these calculations within the server cluster to minimise the effective load. The updating of these filtering rules needs to be done on a regular basis that is environment specific. If the environment consists of fast moving users, the rules will need to be updated far more regularly that if the environment consisted of mostly stationary avatars. Without these filtering rules, the quantity of data being sent to a client will be far more and a trade-off will need to be made as to the cost of the calculation of the AoI lists versus the congestion that could be caused by the data being sent. The main advantage of using this technique is that a client can specify their AoI dependant on their ability to display and receive information. Receiving data from the entire environment is neither realistic nor practical although a user could still do this if they so wished, by setting their AoI to be larger than the size of the entire environment.

- *Level of Detail* (LoD) [Kru01] [Ari99] is another method to reduce the data being received by the user. Consider an environment with dynamic weather and highly detailed foliage. A user not able to display this highly detailed information could request a very low LoD that has only a sky colour representing the weather and cone-like trees. While the LoD can be used to limit what is displayed, it can also limit the data received as a reduced LoD limits the sources of complex data.

- Data limiting, through the *removal of redundant information* caused by update data being available faster than the receiver is capable of receiving it, is a simple matter of time stamping or numbering data in the output queues. If new data arrives for the same object, the older data is replaced. This is particularly useful if clients are being offered the option of receiving data at a rate specific to their needs. Unfortunately, offering this service to the clients limits the use of multicast connections as multicasting data requires all the users in a multicast group to receive all the data sent.

- The smoothing of data retrieval and limiting of data rates using *pre-fetching* algorithms [Ari99] [Par01] is a means to reduce the bandwidth needed over time. The pre-fetching algorithms retrieve information at a slower rate, but do so before it is needed so that the client software does not have to retrieve the information rapidly when it is needed. This could be of significant use in the design of a networked VE as it reduces the maximum bandwidth needed and reduces delays between client software needing information and that information being available.

3.6 Rendering

The experience offered to the user is highly dependant on the output to the user. The rendering of the environment data must be of significant quality to ensure that the user is able to view the environment without breaks in the user's presence with the VE. The rendering needs to be of a high enough quality with research showing that a frame rate of at least 30 frames per second is required for smooth picture interchange. If frame rates fall below this, the user's view can become 'jerky,' thus causing a break in presence.

To ensure that the frame rates remain high, the data that is rendered can be limited. An initial option is to reduce the LoD of the scene. This will either limit the number of objects displayed within a specific scene or limit the detail of the objects displayed [Kru01]. For example, only the larger objects could be displayed, or objects that are more complex could be simplified. A second option is to reduce the AoI as this limits the number of sources of information around the user and thus limits the number of object updates that must be calculated between the rendering of frames. Both these concepts need to be considered and implemented in the design if the design is to be able to support users with differing graphical rendering capabilities and needs.

3.7 QoS Connections

Certain transmission protocols offer connections with a guaranteed QoS. As an ATM network is being used in the design, it is important to use the offered services appropriately. VE connections often have very strict delay and jitter requirements, as noted earlier, and can be RT in nature. These RT connections can be broken into two sub-categories, the first requiring guaranteed data delivery for critical environment data, the second being data that is constantly being updated therefore losses can be tolerated. In addition to the RT connections, the VE can use slower non-RT connections for downloading model information and initialising servers and clients into the environment. Using the connections of an appropriate QoS within the environment is essential and is yet another trade-off necessary in the VE.

By including methods to limit the data sent and thus the bandwidth used, it is possible to offer connections at varying QoS specifications relevant to the users' needs and capabilities. This will ensure that users receive data at a rate that they can process and transmit data at a rate that their peers or the central server is able to receive and process. In addition, this allows a specific environment to set an upper limit on the quantity of data that a user can transmit. This allows the receiver to limit the number of connections it can safely support and allows a load value to be calculated for the node which is representative of the number of connections it can support at the decided upon rate.

3.8 Security

Security within a VE can come in two forms: security for the data being sent and received, and security of identity.

Data security can be simply handled by encrypting the data, although this increases the overheads both in sending and receiving data, as it will need to be encrypted before sending and decrypted when received. The design should include the option of secure data transfer although this will force stricter QoS requirements on the client's connections to the VE and cost the client more in a commercial environment.

The validation of user and server identities is important to ensure authentication and the prevention of mimicking (where a user or server pretends to be 'someone else' to gain an advantage in a particular situation). It important to validate users and servers when connections are made and data is sent within

the VE. Without secure and unique identification, there can be no object ownership within the environment and servers will not be able to join server clusters securely. If point-to-point connections are used within the environment, identification should only need to be checked on connection creation, assuming the security of the channel, while in a broadcast or multicast connected communication, either every packet of information must be checked, or every user must be trusted.

Billing and Statistic accumulation will also be impossible if users within the VE cannot be reliably identified.

3.9 Peer or Server Transparency

As the networked VE expands, so does the number of peer-clients or servers to which a user's client can connect. The proliferation of 'destinations' adds to the complexity of the environment and to the overall confusion to the user should they need to keep track of the peer-clients or servers.

Transparency is a term used to describe the hiding of the complexity associated with the peer-clients or servers. This is done by ensuring that the user need only know a few 'well known' pieces of information that can be entered once and this information will allow the user's client software to locate the servers or peer-clients that form the basis of the VE.

The Domain Name System (DNS) in the IP arena [Moc87a, b] is an example of a service that is offered to hide the IP addresses of servers behind commonly used domain names. While there are specifications for an ATM Name System (ANS) [ATM00b] [Jai00], a specific implementation will need to be considered to offer the services needed to ensure transparency within the networked VE.

3.10 Device Support and Interaction Tools

Support for various interaction devices, as described in Appendix D, is essential to offering a VE that is truly immersive to the user. Similarly, ensuring that the tools are provided with enough data to function correctly is important.

Another aspect of device support is ensuring that users with limited device support and hardware are still able to connect to the VE and that they are given the support, services and data links that are appropriate to their needs. Exclusion of users on the base of limited hardware capabilities severely limits the usability of the VE design.

3.11 Support for Legacy Protocols

To ensure support for a growing networked VE, it is important to allow connections using varying protocols to connect to the VE. While it is understandable that most modern VEs stipulate that you have to use the most modern software, the option of using a mixture of older and newer protocols needs to be considered as it allows for both gradual enhancements to the VE and for an uninterrupted service to the users

Unfortunately, the support for the legacy protocols introduces extra overheads both in the translation processing of the packets received and in the increased size of the data being sent. These overheads reduce the scalability of the environment by increasing the chance of bottlenecks forming. A simpler approach would be to allow the brokering of connections to notify connecting users that they need to retrieve the latest protocol updates.

3.12 Billing and Statistics

The commercial implications of the VE ensure that user statistics and billing are important and need to be kept in the VE. The statistics will need to be secure and identifiable to a specific user. The statistics will enable the design to give the clients better service and to enable the network providers to support the data transfers needed for the VE to function within the delay limits.

3.13 Dynamic Data Management

VEs are often able to supply a fixed graphical package to their users, as is the case in most gaming environments. This is unrealistic in a scalable growing VE as new environment models and other environment data will be continually added as the environment grows and evolves. The management of this data is critical as the models for the entire environment can require large quantities of space.

Virtual Environments cannot simply be rebooted when changes occur. The changes have to be dynamically updatable both in terms of the changes to the environment and in terms of the updating of the users. This links in with the persistency of the environment, which must also remain constant despite changes made from within the environment.

In addition to the dynamic updating of the environment, as the commonly used models can often take up large quantities of storage space, there is a need for the clients to use their storage space

dynamically to store a collection of models that are most commonly used. This limiting of space is essential although this does require that models be retrievable at will from within the environment controlling servers. In addition, on-the-fly retrieval of models can often be time consuming and there will need to be methods in place to ensure that the models are retrieved well before they are needed. This is often called pre-fetching or pre-caching.

3.14 Composite Environments

Virtual environments are not simply a collection of models that can be viewed, but a collection of services that can be offered to the clients. This includes such services as audio or voice communication, media or video streaming, text communication, and many others. Each of these services is part of the VE experienced by the user and a means to include the additional components needs to be offered in the design.

Chapter 4 The DVE Design - HEAVEn

The various functional requirements discussed in Chapter 3 cover various focal points that need special attention when considering the design of a VE or DVE architecture. This chapter builds a DVE architecture, with the described functional requirements in mind, starting with a basic and rather naïve client-server model. While the logical portion of the design must consider the physical networking used, this chapter separates the physical design from the logical application (or session level design) to simply the definition of the architecture. The final design has been named HEAVEn and this is the design that is then implemented, tested and discussed in Chapter 5.

The first section of this chapter deals with the development of a design from a 'naïve' design to a combination of client and server applications. This makes up the application level structure between which communication takes place over logical sessions within the environment. The second section describes the way these logical communication channels translate into physical communication channels that take place over the physical network.

4.1 The Initial Design

The HEAVEn design is described by first presenting an initial and rather naïve picture of a client-server VE. This is then used as the base on which the distributed architecture can be formed. The initial design focused entirely on the application and session layers.

4.1.1 A Naïve Design

A naïve VE design could be as simple as an *environment server* that has a *database* of all the *objects* and *users* within the virtual world, a means to *add*, *alter* and *access* the database, and a means to *interface* with the clients. To achieve the interaction between the clients and the environment database, there needs to be some means of networked communication between the clients and the centralised environment server. An abstract view of this naïve design is shown in the Figure 4.1 below. Note that the environment server has a bidirectional connection to each client and that the clients can only communicate with each other by sending data via the central environment server.

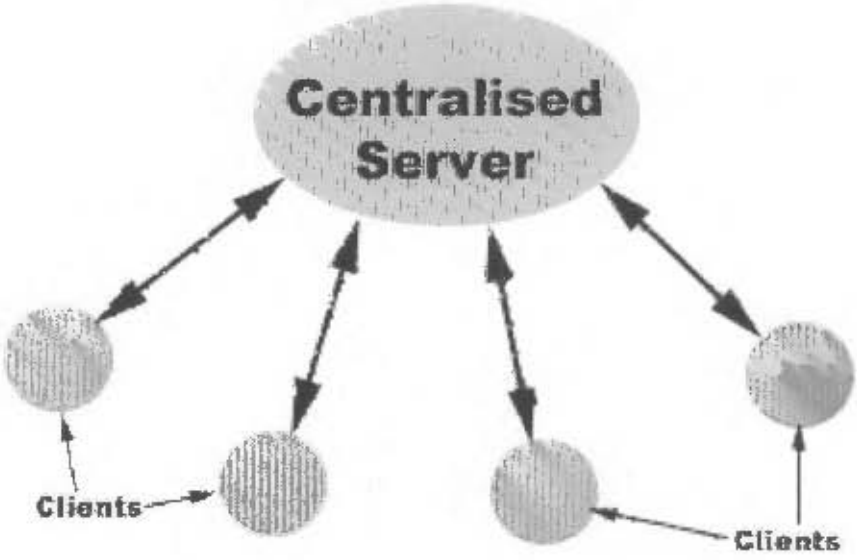


Figure 4.1: A Naïve Client-Server VE

The HEAVEn VE architecture, aimed at supporting a diverse group of users, cannot be as simple as the above design as it does not support or offer the functionality required and noted in Chapter 3. If the aim of the design was to support a limited number of users, this design might have sufficed. The designed architecture must however be scalable in terms of the number of users and the centralised server option will result in a bottleneck, of one form or another (e.g. network bandwidth, processing time or server memory), at the central server[7]. Other design options will need to be considered to counter this obvious flaw in the naïve design.

There are two possible avenues of approaching the next design phase. Either the server can be removed entirely creating a server-less peer-to-peer architecture, or the server can be distributed to spread the load that would normally need to be handled by the central environment server.

4.1.2 Server-less Designs

A server-less design is simply a peer-to-peer architecture where each peer is one of the clients (also know as peer-clients) in the VE architecture. Note that the peer-clients are logically connected to each other but are not necessarily fully interconnected. This is shown in Figure 4.2 below.

---

[7] There are other problems associated with a single centralised server design although the bottleneck situation caused by limited resources is the major factor that limits the use of this simple design.

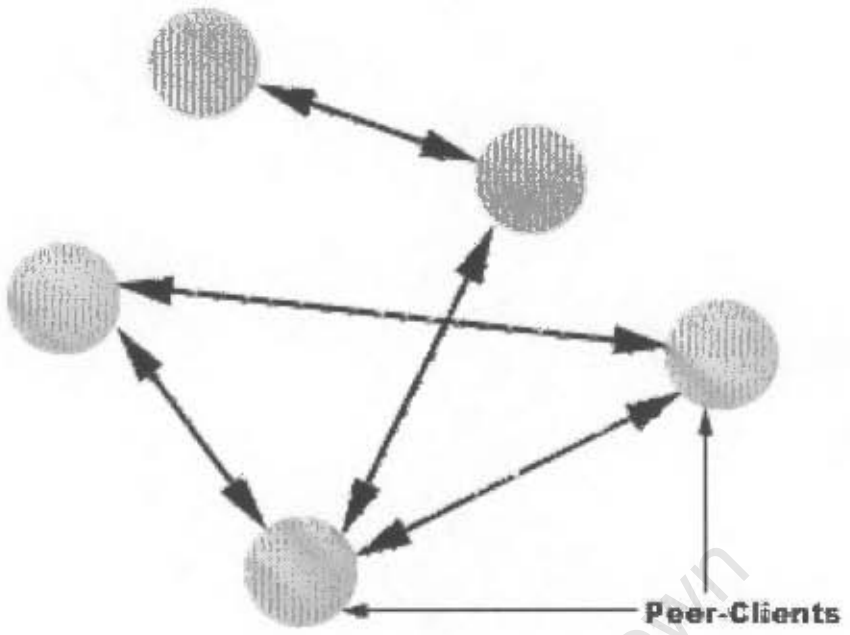


Figure 4.2: A Peer-to-Peer (Server-less) Design

Figure 4.2 illustrates the concept of a peer-to-peer architecture. While this architecture offers a decentralisation of the environment server described in Figure 4.1, there are several significant problems associated with its use. A major drawback is the means by which the peer-clients store and distribute the environment database. In the initial design, the central environment server stored the entire environment database, distributing portions of this information to the interested clients when the need arose. In a peer-to-peer architecture there is no central store and the clients must therefore store their own copies of the environment database that they are using. This can introduce several problems, some of which are:

- *Database Security* - Any individual peer-client will have direct access to the environment database that it stores. This could result in a client altering or selectively sending database information to other clients.

- *Database Consistency* - The peer-clients must either store the entire environment database on every peer-client, which may result in problem situations when updating the database of every connected client (necessary to ensure database consistency), or the peer-clients must store portions of the entire environment database, which may introduce problems associated with the reliability of the information stored in the database as there can be no guarantees made as to the availability of connected clients storing the various data. This will be discussed further shortly.

- *Environment Resources* - The functioning of a peer-to-peer architecture relies on the fact that each element in the network is able to, at some point, interact with the other elements. This is not limited to the communication between the peers, but requires that a single peer-client be available to supply information to any interested peer-client. This data is not just about the user, but can be about the environment or even data used to display the environment. In addition, the peer-client may be called on to route data within the peer-to-peer architecture, as often the network cannot be fully connected, nor to the peers have the resources to be interconnected in this way. As the user base, for which the HEAVEn architecture is being designed, can have widely resources, it cannot be assumed that any single peer-client has enough resources to participate effectively in a peer-to-peer architecture. If the weaker peer-clients participate, they will eventually result in bottlenecks and overall performance degradation. This for both the other peer-clients who are not able to access data at the rate they require, and for the weaker peer-clients' users, who would be unable to experience the environment effectively due to the load on their own resources.

The designed VE must be able to support users who have functional needs that differ significantly. At the same time, these users will have hardware that varies considerably in its capabilities. From those using the environment for research purposes, with high-end workstations, high bandwidth connections, requiring low latency and extreme accuracy, to home-users with low-end consumer terminals that use the VE for socialising, gaming and relaxation, the design must offer the appropriate functionality to the connected clients. The VE must therefore be able to support these varying needs and offer the functionality required by a diverse user group.

It can be argued that a VE design will never truly be diverse and that bespoke systems would suit the research or social needs (using the examples described earlier) far better. This may be true, but the need for an extensible VE, that can rival the World Wide Web for multi-functionality, would ensure that the Internet becomes the fully interactive environment, or *virtual community,* that is so often described. This is clearly not possible in the peer-to-peer network as a single peer-client, regardless of the resources, could be called on to supply a service to the other peer-clients that it is simply not capable of supporting. If this were to happen, the interactive experience of some of the users would be jeopardised. While there are methods to duplicate the sources of this data creating redundant peers, there is still the matter of the temporal nature of the connected clients and there can therefore be no guarantees as to the availability of the data as would be a requirement in the example of a research facility.

Further, when considering a peer-to-peer VE where each client requires a portion of the environment, each peer-client must store either an entire copy of the environment database, or an appropriate portion of it, to allow the client to render and display the area in which the user is interested, interacting with and interested in. If only a portion of the database is stored at each peer-client and information is required by the peer-client, this data can be requested from those peer-clients that already have the required information. This requires that there be communication between the peer-clients to enable the location of the information to be published, which increases peer-group communication. It also creates a situation where a specific peer-client or a group of peer-clients can become critical to the environment consistency as disconnection of a specific peer-client that is storing critical data will result in that data being unavailable to the other peer-clients. This problem can be solved in two ways, either by entirely replicating the environment database across all the peer-clients, or by replicating portions of the environment database so that there is sufficient duplication to ensure that peer-client disconnections do not affect the overall consistency of the available database.

4.1.2.1 Partial Database Replication

When portions of the environment database are distributed to the various peer-clients, the architecture must ensure that there is *sufficient* duplication of the data to prevent data loss should a peer-client disconnect from the environment for any reason (including failure). While there are algorithms that attempt to share the data and distribute it sufficiently [Li89] across the entire set of peer-clients, unless the data for the entire VE is duplicated in every peer-client, only statistical guarantees ensuring database consistency can be made. This also assumes that the network connecting the clients remains stable and active. If portions of the environment become unreachable, the data is similarly unavailable to the rest of the environment and the duplications algorithms will need to ensure that the duplication is not just statistical distributed, but geographically distributed as well. Irretrievable loss of environment data in these manners is particularly destructive to an environment that is attempting to ensure the consistency and believability of a VE.

Solutions to this situation could include the duplication of all the data to all the peer-clients, or the adding of peer-clients that are persistent and store a complete copy of the entire VE database and from which all other peer-clients can gain the needed information. The second option is simply a hybrid client-server approach where there are multiple servers and where the clients can gain database information for clients as well as servers. The first option is of more interest to the server-less design approach while the client-server concept will be focused on in the next section.

4.1.2.2 Complete Database Replication

If the entire environment database is duplicated in every peer-client, then there should never be a situation where a disconnected client causes the loss of environment data that affects the entire VE. The problem that must now faced is that of ensuring that the database stored in each client is consistent and remains so, despite the many changes that can occur in large-scale VEs. If the environment database is to remain consistent, then every client must receive every update to the environment or must receive regular updates that summaries these changes. Even if the peer-clients use algorithmic schemes to compress, limit and filter these updates, the quantity of data will rapidly increase as the size of the VE grows. If multicast or broadcast communication is used to limit congestion and bandwidth usage in the network, the congestion at a receiving client will still increase with the size of the environment and very rapidly swamp the client's network connection. Since we are considering the case where clients have differing capabilities, we cannot assume that any or all of the clients are able to support this system and the design must be able to support the 'weakest link' rather than forcing every client to maintain a complete and consistent database of the environment.

Scalability and the use of the VE by a broad range of users are two areas where the HEAVEn design is being aimed and the design can therefore not rely on the complete duplication of the VE database as this limits the use of the environment. The clients can therefore only be required to store data regarding the portion of the environment that the user is *interacting* with and *interested* in. The storage of the environment database and updating of this database occurring through user actions and interactions must be done using dedicated environment elements or servers.

Limiting the processing, storage and general requirements for the client-side system is a trend in modern computing as both hardware and software vendors realise that forcing their user to upgrade to the 'latest and greatest' hardware does not endear the software supplier or their software to their users who must continually pay for this newer hardware. Instead, the services must be offered to the users at the level their equipment can handle. If the option of upgrading their hardware is chosen, any additional functionality that may now be available should automatically be used and in this way, the vendors encourage the users to upgrade but do not force them.

The choice of whether to use a peer-to-peer or client-server design is thus relatively simple, as user hardware can't be relied on to provide a consistent environment database, whether distributed or not, and it is therefore necessary for the designed architecture to include specific hardware to compensate

for the shortcomings of the users' hardware. This aside, both peer-to-peer and client-server architectures have advantages and disadvantages and thus a hybrid architecture which combines the best of both concepts, must be developed. A closer look at the advantages and disadvantages of the peer-to-peer and client-server architectures can be found in Appendix B.

4.1.3 Distributed Server Designs

In the naïve design, shown in Figure 4.1, the central server receives update data from all the connected clients about their user's interactions with and within the environment. The server can either forward this data blindly to all connected clients or intelligently filter, combine, alter and check the data before sending it to a subset of the entire client base. The clients are selected based on their interest in the data, the privileges of the client to view the data, and client's ability to display and receive the data. This concept is particularly appealing as it removes significant load from the clients, as they do not need to process unwanted data. However, this load must be taken up by the dedicated environment servers and while they are dedicated, this load will rapidly grow as additional users join the environment. As with any single server, no matter how powerful its hardware, it will eventually become a bottleneck and prevent the further scaling of the environment. To prevent this situation, and enable further expansion, the central environment server must be distributed into what is known as a *server cluster* or *federation of servers*. The questions that must now be asked, and will be answered in the following sections, are:

- How will the environment database be partitioned across the servers? Must it be partitioned at all?

- How will the distributed servers be connected to each other?

4.1.3.1 Database Distribution

The first question posed has several possible solutions. The database can be divided according to the logical areas within the VE, divided according to physical disk limitations, or not dividing but duplicating the entire database allowing each server to distribute similar data to a portion of the connected users thus the division occurs across the number of connected users.

When considering that the design must ensure scalability, any solution that forces a client to connect to a *specific* server to retrieve data is actually creating a number of single server scenarios grouped together to form an environment. These partitioning schemes would result in the environment failing if a portion of the environment, controlled by a specific server, became too popular for the server to support the number of clients connected, regardless of the availability and load of the other servers in the environment cluster. Similarly, if a server controlling a portion of the environment was to fail, that entire portion would be lost and this would seriously jeopardise the consistency of the environment. When considering that the consistency of the environment is of prime importance, it limits the possible schemes for dividing the database to those that divide the number of users across the available servers according to the number that each server can support without the server's service to the clients being degraded[8]. This also ensures that the servers can be of differing capabilities and adds to the more generic and scalable nature of the designed architecture.

That leaves the solution of a complete duplication of the environment database across the entire cluster of servers. To maintain consistency, this design choice requires that all database update information be sent to *every* server in *every* cluster so that the servers' databases remain consistent and that clients connected to other servers can receive the update data if they require it and are interested in it. As the updates between the servers must ensure consistency, the updates cannot be lost and must therefore be guaranteed.

The design does, however, allow each server to respond to client's requests for data about the environment rapidly as the data is local to the server. This data does not need to be checked (as it is already valid) or fetched (as the entire database is duplicated), although conflict situations (such as a request to destroy an object) must be negotiated between the servers. This ensures that no other server or servers are attempting similar actions at the same time (such as two avatars attempting to grab an object at the same time).

Another advantage of this design is that users can be directed to connect to a server that is geographically *local* to the user. This reduces latency times between the client and the server, as the users' connections tend to be of a lower quality than those used between servers. Similarly, the clients

[8] One can consider a database division according to logical environment boundaries where each logical portion is supported by multiple servers in a mini-cluster. This concept, while valid, is simply a collection of the user-divided architectures where the database 'portion' is seen as an environment of its own.

will only be charged for the geographically short connection to the local server and the quality of service offered by the connection will not need to be as strict as it would be if the connection had to travel geographical further to reach the server, thus further reducing the cost to the user.

When only considering the servers, it is important to note that the size of a typical VE database can be large if the database stores all the model, video and audio data that can be used in the reproduction of the VE. If the environment servers store only the data describing the actual environment, the servers will have to store far less. The media data can either be stored in environment specific stores where clients can connect and retrieve it when they require it, allowing for a dynamically alterable environment, or the data can be preloaded by the user, which is the common scenario in current VE design strategies. The data stores will be discussed later.

4.1.3.2 Server Interconnections

The second question can be answered by either connecting the servers in a peer-to-peer or hierarchical manner. The hierarchical connection structure is discussed further in Appendix B but will be summarised here. The hierarchical structure allows the environment, which we have decided is best fully duplicated, to pass received data to a higher level where it can be collated and passed on to other servers. While this collation can reduce the quantity of data that is being distributed to each server, this process will increase the latency in the design. These latencies are introduced at the higher-level servers where it must be received, processed and collated before being sent, and these processes will take additional time. The extra delays are not strictly necessary, as the data could have been passed directly between the servers. Due to the RT nature of much of the environment's communication, these delays could seriously hamper the scalability of the design. Limiting the latency between environment entities is one of the highest priorities for the HEAVEn design, as any introduced latencies will limit the design's scalability since users without faster connections will not be able to maintain the minimum latencies required for VE interaction. The higher-level hierarchical servers also introduce extra central points of failure. The aim of distributing the servers was to eliminate this situation and the use of a hierarchical infrastructure to interconnect the servers simply reintroduces this problem.

The system of peer-to-peer server interconnections must therefore be chosen as this option limits the latency in the design although this does create a trade-off between the number of interconnections

between servers in the server cluster and the bandwidth used between these servers. The choice of peer-to-peer server connections offers two options, partial interconnection or a fully meshed[9] design.

The partially connected design must assume that there will be cases where the source and destination are not directly connected. This will result in situations where data has to be routed through peer-servers and this clearly introduces extra delays. These extra delays result in a similar scenario to that described above.

The fully meshed option ensures that if a source server sends an update, every other server will receive the update without any intermediate servers having to handle the data. While this does not ensure that the data will be received in the shortest time, if the transmission times for the connections between the servers have a maximum bound, the overall system's communication latency can have an upper bound guarantee. When considering the design of a fully meshed system, it is important to remember that every node must be connected to ever other node in the mesh. The number of point-to-point connections in the mesh will grow very rapidly if individual connections are used. Focusing on technologies that allow multicasting is therefore important.

The advantage that using a multicast technology offers in this scenario is that if the servers are connected via a multicast group, every server will receive the same information and at minimal delay costs. ATM natively supports the use of multicast technologies and at the same time, ATM also offers various guarantees to the data being sent and this can be used to ensure that any maximum latency requirements are met. Alternatively, there are multicast frameworks [Sat98] [Sat00] designed for use in IP networks although the overheads associated with the guaranteeing of the data being transmitted, which ATM offers natively in terms of latency, bandwidth reservation and data ordering, prevents these frameworks from scaling to multicast groups with a large number of participants.

A *fully meshed* architecture assumes that any point in the mesh can send to every other point in the mesh. This is a 'many-to-many' relationship and must be available at the application level to ensure that communication can take place between all the nodes in the server mesh. While many-to-many

[9] A fully meshed design ensures that every server node is directly connected to every other server node in the mesh. No interconnections pass through any other server nodes other than the source and destination. While this is the logical view of the architecture, the physical view of a meshed design can consist of multiple network nodes between source and destination although each pair of server nodes will have a direct connection between each pair of servers.

connections are not supported directly in the ATM Forum's *UNI 4.0* [ATM00] specifications, it is important to remember that there are currently available and planned extensions to these ATM specifications that offer many-to-many multicast connections that still maintain ATM's guaranteed QoS [Leo98] [Tur97].

Assuming that these extensions are not used, multiple single-source multicast connections can be used. Each server would therefore create a multicast connection to every other server with itself as the source. The main drawback with using this option is that a multicast connection is used by every server in the environment cluster. One of the main aims of the HEAVEn design is to make the design usable over a non-dedicatee network. As many older ATM switches do not support more than 256 multicast connections, the scalability of the environment cluster could be limited by this requirement because the full 256 multicast connections may not be available to the HEAVEn architecture, as other applications may require some of them.

If the situation arose where no multicast connections are available, a connection between every server would have to be created. While this is possible for a small VE, the scalability of the environment would be seriously hampered as the addition of new connections is of $O(n^2)$ algorithmic complexity. This compares with the addition of a new server to a list of one-to-many multicast groups which results in an algorithm of O(n) complexity, and the addition of a new server to a many-to-many multicast group which is of O(1) complexity. The lowest complexity is clearly the option that would be most preferable as any additional work done by the servers that is not aimed at offering a better service to the clients, increases latencies in the response times of the servers and could jeopardise the client's interactive responses.

4.1.3.3 Aspects of ATM Multicast Connections

The ATM multicast connections offer several integral services to the connectivity that is being used by the VE. As described more fully in Appendix C, these connections remove the duplication of data in the ATM network path and significantly limit congestion by only duplicating data in the network if the paths along which the data must travel, diverge. In addition, certain connection characteristics, such as the maximum latency required to ensure user interaction, can be guaranteed using a QoS contract that is set for the entire multicast connection.

The delivery of the data in an ATM network is never completely guaranteed. If the data arrives at the destination when using an AAL5 connection, it is given that the data that has arrived is correct, while if

the data does not arrive, it must be resent or ignored as appropriate. As portions of the environment data are critical to ensuring the consistency of the environment database across all the servers, the delivery of this data must be completely guaranteed. This can be done by either using a fully reliable multicast connection protocol or an application level acknowledgement protocol. There are extensions to the ATM 4.0 specifications that offer reliable connections [Tur96] although the use of these extensions cannot be assumed in a non-dedicated network. If data delivery must be ensured, the best option would therefore be to include an application-level acknowledgement protocol. This will unfortunately reduce the usefulness of multicasting as the acknowledgement data packets would have to be transmitted over dedicated connections as only the sender needs this information and distribution over the inter-server multicast connections would distribute the acknowledgements to all the connected servers and be wasteful of bandwidth and cause server extra congestion. It is therefore important to consider if a multicast connection is truly useful in the situation or if multiple point-to-point connections should be used.

Another aspect of multicast connections is that a single multicast connection has the same QoS specifications for all connected leaves (in this case, the servers) in the multicast tree. This can be a problem if the destinations require differing quantities of data but in the situation where all the leaves require the same data (in this case, to ensure database consistency) it is not a problem. In fact, the generalised QoS is a means to ensure that all connected servers maintain a standard QoS between them and this ensures that all the servers in the server cluster remain within the interactive response time bounds.

The architecture, as currently proposed, is shown in Figure 4.3. Points to note are the meshed server cluster that allows clients to connect to a virtual centralised server.

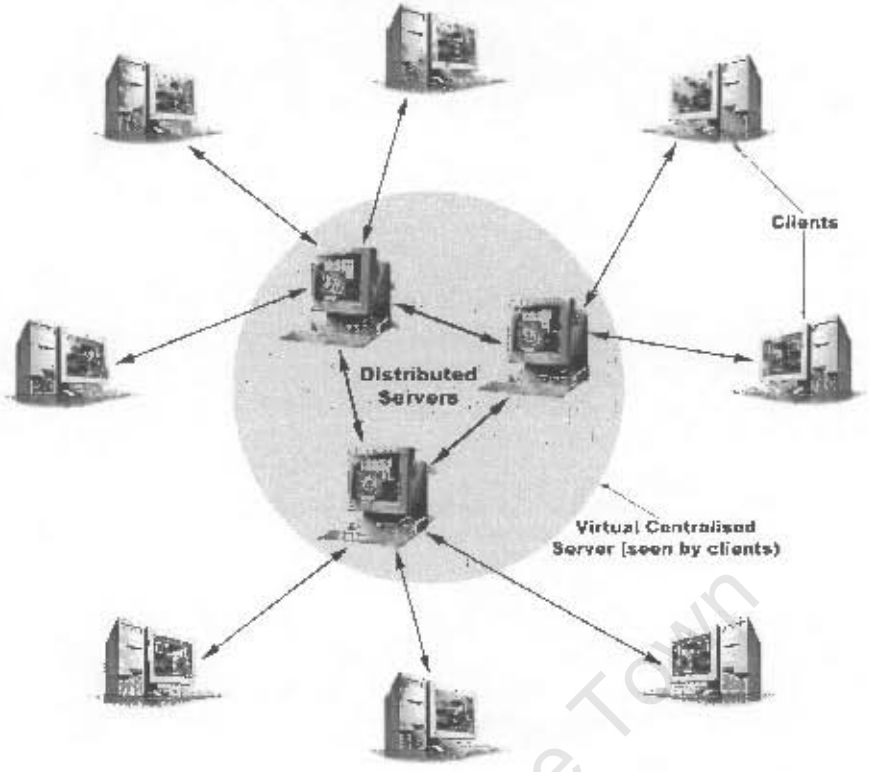


Figure 4.3: The Design Including the Distributed Environment Server Cluster

4.2 Realising the Initial Design

While the design shown in Figure 4.3 might seem rather simple, several aspects are useful although there still needs to be a great deal of design development before a client can transparently connect to the server cluster, retrieve environment data and interact with other clients in RT.

4.2.1 The Logical Centralised Server

While scalability, consistency and end-to-end latency are crucial aspects of a VE's design, when distributing servers, it is important to ensure that the logical client-server metaphor remains consistent and that the complexity of the environment servers is transparently *hidden* from the clients. To enable this, the architecture must provide a lookup service that will transparently direct connecting clients to an appropriate environment server (or in the case of distributed environment servers, to a group of environment servers that supply the client with necessary services).

This lookup service is supplied by a *Broker Server* (referred to as a *Broker* hereafter). The broker accepts connections from connecting client software, authorises their connection request, supplies the

client with a unique environment identifier and then directs the client software to connect to appropriate environment servers within the environment cluster.

Similar to the reliability problems associated with having a single server, the VE cannot rely on only one broker to serve the entire environment as this could result in a bottleneck situation or a single point of failure for the entire VE preventing *any* new connections. To alleviate this problem, multiple brokers must be used, which must be interconnected to ensure that updates to the brokers' user-database and server-database are sent to all the active brokers, thus maintaining the consistency of these databases. The interconnection between the servers must also allow negotiation of conflict situations, such as two users requesting the same user ID at the same time, further enhancing the consistency of the VE.

A fully interconnected mesh is therefore used as it allows all the brokers to simultaneously receive and send updates to all their peer brokers. As the communication must be reliable, the communication between the servers needs to be guaranteed and the use of a simple acknowledgement protocol[10] is essential. The fact that acknowledgements are sent will result in data inefficiencies if multicast connections are used as the acknowledgements only need to be sent to the source of the data and not to the rest of the broker mesh. Using multiple point-to-point connections would therefore be the best option as brokers can thus reply to and acknowledge received data directly. The use of the acknowledgement protocol ensures that conflict situations can be avoided by ensuring that all the brokers agree to an action when it occurs.

A new environment server wishing to join the environment cluster also needs a means to locate the cluster before it can attempt to join it. The brokers supply this service and offer a list of active environment servers to the connecting server. Once the new server has been authorised by a broker, and it has joined the environment server mesh, it contacts a broker to register so that it is active and that the brokers can begin directing new clients to it. The broker contacted will propagate the registration data throughout the broker mesh to update the other brokers. Similarly, servers leaving the environment cluster need to deregister with a broker before leaving, allowing the brokers to remove the server from the active server list and to propagate this information to all other brokers.

[10] The acknowledgement protocol does not need to be defined here as a standard positive acknowledgment algorithm can be used.

As the brokers store a list of environment servers, it is important to ensure that the servers are *active*. There are two ways that an environment server could leave the environment server mesh: through controlled departure or through failure. The first option allows a simple deregistration from the brokers, as described above, while the second could result in an inconsistency in the brokers' lists of active servers. This is prevented by the brokers regularly requesting activity updates from the environment servers in their active lists. The updates received are distributed to the entire broker mesh to ensure that each broker does not independently request updates from every environment server (adding to server-side congestion) and to maintain a consistent environment server database, across all the brokers, from which clients can request information.

Brokering the connections to the environment server cluster has several other advantages:

- As a broker deals with *initial connection administration*, the activities of the environment server, to which a client connects, are not significantly affected by the addition of the new client. The registration of new clients, connection fee calculation, user identification, password authentication and authentication token distribution are handled by the client's broker instead. This is extremely useful as even small delays can adversely affect the performance of the environment servers.

- When clients request connections to the environment as a whole, the broker *authenticates* their username and password before returning an encrypted-token that can be used when connecting to the environment servers. The environment servers will therefore only need to check the client's authentication token to ensure that the connecting client is secure. For increased security, the environment servers can connect back to a broker to validate the client's token, although this will increase the impact of a connecting client on the environment server, which will have to wait for the response from the broker. This is a trade-off between the environment's security and the effect that any interruptions could have on the currently connected users.

- The *activity requests* to the environment servers can be used to request other, server specific, information such as the server's *Load Index*. The load index is a value reflecting the server's load in terms of network usage, processing power, memory usage and similar server limiting factors. The choice of a formula to combine these factors is an environment specific choice as

a VE with large memory overheads will need a load index that is weighted towards the memory available while another VE, which may require more processing power would have a load index weighted towards the free processing time available. As long as all the environment servers in the environment cluster use the same method to calculate their load index, a comparison can be made between all of the load index values and the brokers can then direct newly connected clients appropriately to *balance the load* on the available servers.

- In addition to the distribution of the client load across the available servers, the brokers are able to monitor the overall environment cluster's load. If the average load index reaches a threshold point (again, specific to the environment cluster), no more users will be allowed to join the environment. The brokers will attempt to initiate new servers that will then join the environment cluster and allow new clients to connect to them. While this does limit the availability of the environment to new users, it ensures that the services offered to connected clients are not compromised by the *overloading* of the environment server cluster.

- When an activity request is made, the connecting broker can also request information about the various users connected to that environment server. This information can be used to collect *user connection statistics* and used for *billing* if the environment is commercial in nature.

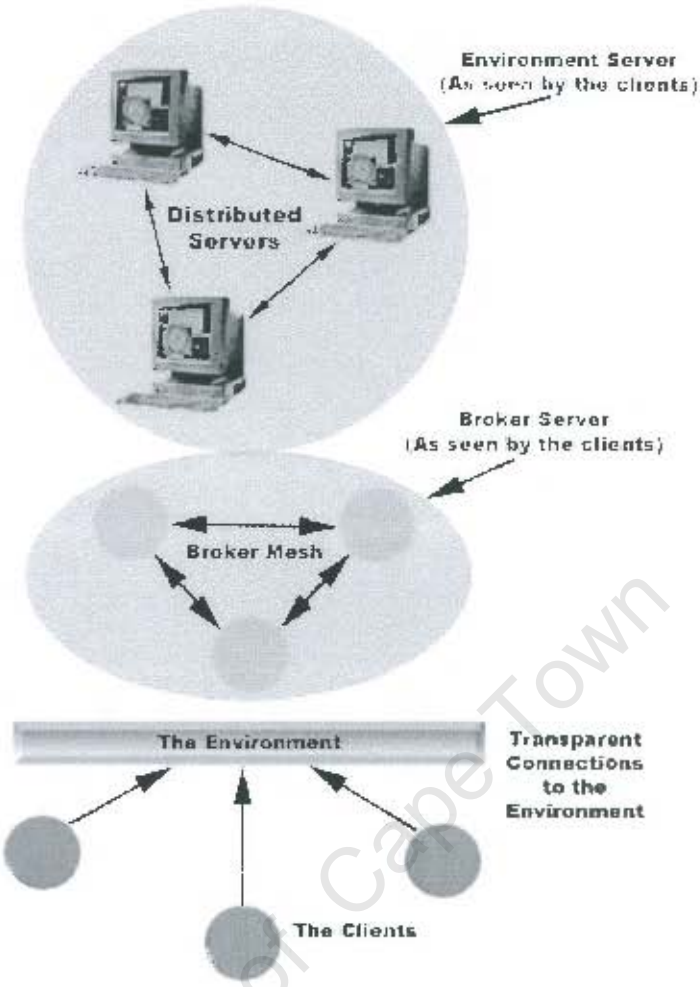


Figure 4.4: Design showing Clients, Environment Servers and Broker Servers

The current design is shown in Figure 4.4. It includes the meshed brokers (using multiple point-to-point connections to allow data delivery acknowledgements), the environment server cluster and the connecting clients. Note that the clients do not see the servers that are the environment itself but only the environment as single entity to which they can connect and behind which the brokers and environment servers are hidden. A connecting client connects firstly, at this stage of the design process, to a broker server that in turn, directs it to an appropriate environment server within the environment server cluster, hiding the complexity of the environment server cluster completely.

4.2.2 Finding the Broker Servers and Hiding Address Information

The means by which the clients locate the brokers has not yet been dealt with. While this might seem like a repeating problem since any new server created to distribute address information will itself need to be located, it is noted that the brokers are *environment specific* servers, as shown in Figure 4.4, and store only the address information of the servers in its environment. A generalised environment non-

specific address distribution service needs to be developed or used. Further, a means to translate a unique '*descriptive environment name*' into an appropriate list of addresses to which the clients can connect, would make locating an environment far simpler. The process of converting a descriptive name into a list of addresses is known as *Name Resolution*.

Name resolution services can be found in the IP domain and are known as the Domain Name System (DNS) [Moc87a, b] and in the ATM domain as the ATM Name System (ANS) [ATM00b] [Jai00]. The currently available implementations of the ANS do not offer the ability to translate a single descriptive name into multiple ATM addresses and it was therefore decided to include a name resolution service that does this in the HEAVEn architecture. The Name Servers (NSs) function as follows:

- The NSs are associated with 'well-known' addresses that will remain 'constant' despite changes in the network or the services offered to the users. Addresses of local NSs will be entered, often into the operating system itself or into the application if the operating system does not support this functionality, so that the NSs can be accessed by applications when they need to resolve descriptive names into address information.

- To enable the NSs to propagate update information to other servers they can be interconnected or can interconnect. While this obviously increases their usability, they can function as standalone servers that server address information. Often the NSs are connected in a hierarchically format to facilitate database segmentation and information propagation.

- In addition to simply storing a list of descriptive names and addresses, similar functionality to that described above for the broker servers can be included into the implementation of the NS implementation to allow them to poll the various stored server addresses to check their activity. Returned information could include the addresses' generalised load index and the NSs could use this to balance the load associated with the various addresses referring to a single descriptive application or environment name. This would also allow the NSs to distribute only the 'active' entries rather than transmitting redundant address information.

- Communication between NSs must be guaranteed to prevent errors in the name database and to ensure that the data being received by the requesting client is correct. This is done by using a simple positive acknowledgement system.

Research into the nature and services offered by a name distribution service is not a focus of this dissertation. The main focal points have been covered in this summary as the service is required for the correct functioning of an implemented test-bed and is one of the implemented services.

4.2.3 Downloading the Multimedia Data

As noted in the section about broker servers, part of the reasoning for having a broker is to reduce the load on the environment servers associated with authenticating connecting clients, as this could hamper the services they offer. In a similar manner, the task of *distributing* and *storing* multimedia data must be removed from environment servers as it is not a time critical function and often requires significant overheads. In addition, the size of all the model and media data is often too great to be stored on a single server and so duplication of the entire database to each of the environment servers is not an option.

The retrieval of the model and media data can be done separately as the environment servers' main task is to store the *dynamic state* of the environment and the model and media data is most often *static*. Even the dynamic models, video or sound files often require only a simple state value to be stored that references a specific portion or place within the file. The model and media data can therefore be retrieved on-the-fly, or pre-fetched [Ari99] [Pop02], and cached before it is needed. Using a local copy of the data will simplify the client's tasks and speed up the client's ability to interact with the user and with the environment as a whole. It is therefore preferable that the client caches retrieved data and pre-fetch needed data so that the client does not need to wait for it to be retrieved. Often clients are populated with a basic set of commonly used model and media files on installation. The cached files will add to the set from which the client can draw its needed information.

The retrieval of the data that is not directly available from the client's local cache is therefore handled via a set of servers dedicated to the transfer of large quantities of data. The servers facilitating this data retrieval in the HEAVEn architecture are known as the Model and Media Servers (MSs). The MSs store the various model and media files required by the clients to render a display of the environment and clients can then request the data from the MSs when they require it.

The choice of the how much model and media data to retrieve and cache, how soon to cache it, and for how long it should be cached is entirely client specific. If there is ever a case where a model or media data source is not available, the client can always insert a 'placeholder' icon within the environment to notify the user that it is still retrieving data while it requests it from a MS. This is analogous to the way our current web browsers insert a temporary picture icon into web pages while the actual picture is being downloaded.

As a single MS cannot be expected to store all the multimedia information for a specific environment due to the size of this data, multiple MSs are required. This creates a situation where a specific server could become a source of failure and it must be ensured that there is sufficient duplication of data across all the MSs to statistically prevent this. Unlike the simple peer-to-peer client situation, the overall trend for the MSs is to remain online and thus they will generally remain active while the clients cannot guarantee that their users will remain online. Further, the loss of a MS server with unique data is still not a critical failure for the overall system, as the environment will continue to function with the clients requiring the data simply using a placeholder instead of the actual item within the environment.

The brokers are important to the MSs as they ensure that a MS does not become a point of congestion. They do this by keeping a database of the multimedia files required for the environment, on which MS these files are stored, the load associated with each MS and the popularity of the various files. Should a specific file become popular, the brokers can initiate inter-MS file transfers that distribute these requested files and thus the load associated with client file retrieval.

- The brokers store a database of the media files required for the entire environment and store where each of these files is available. This enables the brokers to direct clients to appropriate MSs when requests are made for specific information.

- As the requests for specific media files can be logged and counted, the brokers can ensure that there are sufficient copies of the more regularly accessed files at appropriate locations. This attempts to prevent a MS becoming a central point of failure and further, prevents the congestion that will inevitably occur at a heavily used MS.

- Because the broker servers also store the load of each MS, they can limit access to a specific server by directing a client towards a more lightly loaded MS when a client requests the location of a specific file. This must occur before any one of the MSs becomes a point of congestion. As retrieval of the multimedia data is not a RT transaction, this can be done without jeopardising the consistency of the environment and the clients will therefore need to ensure that if they do not yet have the appropriate files needed to render a scene, that they are able to display an 'in progress' icon, model or other identifying symbol.

- If a MS becomes a point of congestion or the brokers find that the files are not sufficiently distributed across the range of available MSs, the brokers can initiate an inter-MS file transfer to redistribute sought after files. The choice of which files need to be duplicated and how much duplication is necessary, is not focused on in this dissertation, although the topic is one for further research.

As the MSs are part of the environment cluster, they must register with a broker to ensure that they are known and that clients can be directed towards them. The registration of the MS includes the uploading of a list of unique identifiers for each file stored. Any alterations to the files stored on the MS results in an update being sent to a broker to notify it of a change of database and the resultant updating of the broker mesh.

The MSs are individual servers and do not require continuous connections between the servers. Any communication between MSs occurs when needed and is used only when transferring files to ensure data duplication and load balancing.

The MSs can be accessed at any time assuming that the requester's security token is valid. This token can be checked by connecting to a broker and validating the token, a task than can be done by any of the environment servers. The request for a file can then proceed, with the client having requested a file using its unique identifier. The MS will then locate the file using a table listing the identifiers and the location of the appropriate file stored. Note that this allows MSs to store their files in different physical locations on their local disks.

The MSs' inter-server communication, communication between the MSs and the brokers and the communication between the MSs and the clients must all be guaranteed as losses in the communication

could cause file database inconsistencies and interrupted file transfers will damage the transferred media and model data. The communications use a simple positive acknowledgment system to ensure that data is received correctly.

4.2.4 Streamed Model and Multimedia Data

In addition to the downloading of model and media data, the MSs can be used to 'stream' data to a client. Streamed media allows the client to receive portions of the data in such a way that the portion can be used once it has been received and without waiting for the entire file to be downloaded. The remainder of the file *adds* to the already retrieved portion. This is commonly the case for video or voice data, portions of which can be used (heard and/or viewed) before the entire data file has arrived, but also for model files where simpler model information can be initially retrieved and displayed, with the more complex model information arriving at later stages in the download process. This can be further extended to assist the clients in the display of models that are of lower detail [Wan97] so that the client software is able to display the models at a lower detail level by only using a portion of the model data retrieved. Without this facility, the limiting of the model data being displayed would not have been as simple a task, as the client would have had to intelligently decide what portions of the composite model to discard before displaying it.

4.2.5 The Clients

The client software, being the main interface with the user and with the environment itself, has several important roles to play and several purposes that must be considered, as they affect which services must be offered by the environment and the environment server cluster.

- The client processes information describing the environment that the user, or the user's avatar, is moving in and interacting with. The information describing the user's surroundings is often called the *scene description* and is a portion of the entire environment database. The description does not necessarily include media data. Since clients may already have the media data, it is redundant transferring the data along with the scene description and the HEAVEn design does therefore not include any media data in the scene descriptions. The scene description refers instead to unique file identifiers that reference specific data files stored on the MSs. These files can then be displayed appropriately as described in the scene description. Note that while no media data is included in the scene description, data describing how the media is situated in the environment and the status of the media is included.

As noted earlier, the media files are not often static and will contain more information than can be displayed at any one time. A simple example of this is a movie file where only one frame is displayed at any one time and a dynamic state shows which point in the movie is being viewed. A model file could similarly contain information about varying poses, such as a human avatar that could be 'standing' or 'sitting', and require a status position to stipulate which pose is being used.

The information about the state of the media file must therefore be transferred to the client by the environment servers while the actual media files can be retrieved on-the-fly, or through pre-caching and pre-fetching techniques [Ari99] [Par01], from the MSs. The appropriate use of these techniques allows the client to retrieve the media data before it is needed or if it cannot, an appropriate 'placeholder' will need to be inserted and displayed in place of the actual data until it is retrieved.

- The rendering of the scene describing environment combines the received data and the media files that are available. If the rendering engine is not capable of completely rendering the entire scene in the allotted time, assuming the client draws between 30 and 60 frames a second, a choice must be made to lower the detail level rendered or lower the quantity of information being displayed. This will need to be communicated to the environment servers so that they can limit the quantity of data being sent to the client. Note that receiving the data and not rendering it is wasteful of bandwidth and processing time on both the client and the servers.

- Once the environment is rendered, in terms of displaying the rendered environment and providing audio and other feedback, the client must accept data from the user in the form of tracked updates, audio, video and other sources that must be processed and sent to the environment servers. The client must ensure that they process enough information to ensure that any user interactions are correctly translated and transmitted to the environment servers to ensure that the interactions remain interactive and realistic.

This functionality forms the basis of the client application. In response, the environment servers must offer data and services to the clients ensuring that it occurs in RT and that the data is consistent across

the entire VE. This will enable the client to offer an immersive, interactive and consistent experience to the user.

As noted earlier, the client cannot be forced to carry too much of the load associated with the VE due to the fact that the user's hardware and connection capacity may both be severely limited and already stressed by the rendering of the environment itself, the processing of the update data and the sending and receiving of the environment data specific to the user.

Further, reliance on the client must be kept to a minimum as the client is not secure and any functionality that is needed for consistency must originate from the servers. An example of this would be the broadcasting of a 'death' message when a user's avatar is shot. This must be considered a 'secure' function as you do not want the dead players 'forgetting' to notify other users and continuing to interact in the environment as if they were alive!

The design of the client software is being focused on in other research [Doy03+] being conducted at the University of Cape Town.

4.2.6 Summary

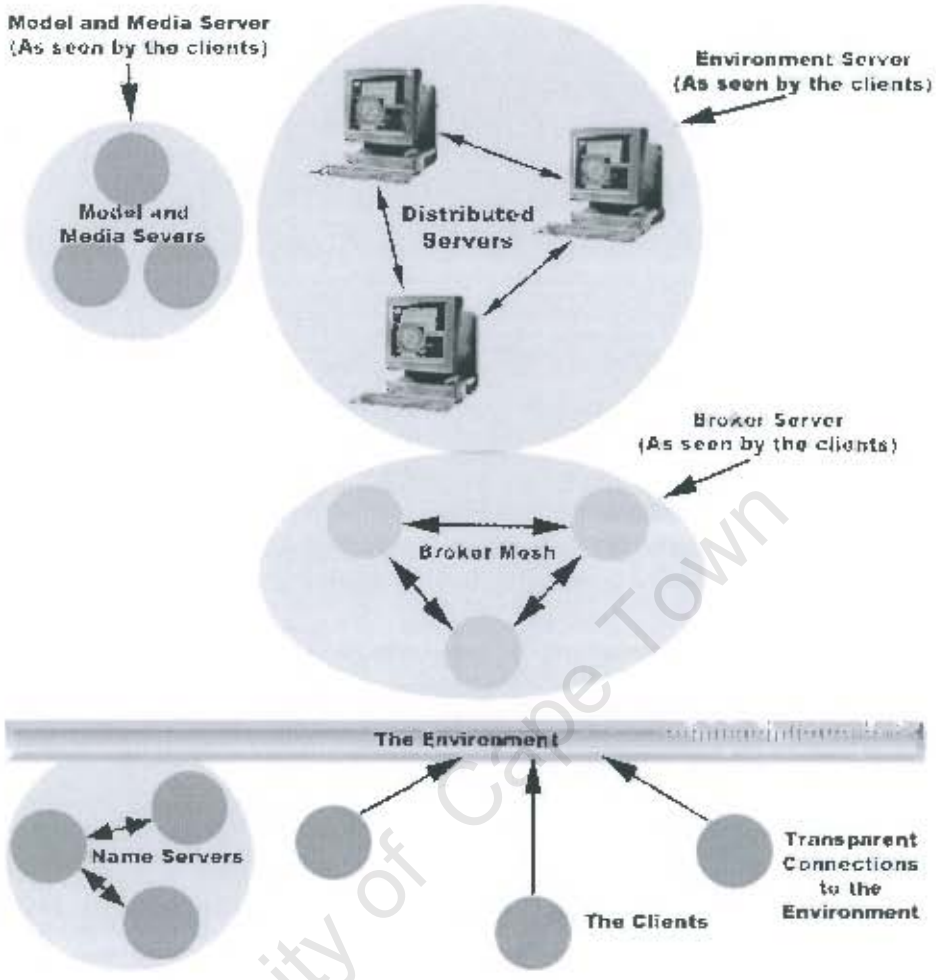


Figure 4.5: The Design Including Model and Name Servers

Figure 4.5 shows the current state of the design including the individual Model and Media server, the meshed Broker Servers, the partially connected Name Servers and the remaining Environment Cluster, which will be the focus of the next section. Note that the MSs are transparently hidden behind the brokers and are standalone although they can communicate with each other as needed. The name servers are shown as being partially linked and are not part of the environment although they do offer services to the clients and to the environment as a whole.

4.3 The Environment Server Cluster

While the initial design of the environment cluster might seem rather simple, several aspects still need to be clarified. This section focuses on the environment servers, the methods they use to communicate with the clients, and a means to ensure that the communications occur with a 'realistic time.' Note that up until this point in the design process, the assumption has been that the environment servers

distribute all environment data to the clients although it was mentioned earlier that the centralised environment server could filter this data. If the choice is made to filter the data, processing time in the centralised server will need to be used to read any incoming data and decisions made to filter the data. These decisions are based on what as inside the received data, so that it can be filtered before being sent to specific users that require the data due to their interest in the source or content of the data.

4.3.1 Consistency, Latency and Data Limiting

Now that mechanisms are available to ensure the hiding of the distributed nature of the environment's servers and the balancing of the load between the various servers, the issue of scaling the centralised server must be focused on. Creating a server cluster instead of using a single centralised server creates several significant problems that must be dealt with in the design:

- The *consistency* of the environment database across distributed server must be maintained.
- The communication *delays* between clients, introduced by the addition of the servers, must be minimised and a maximum delay bound must be set and maintained.
- The *quantity of data* that is being distributed between the servers, and between servers and clients, must be limited or filtered in some way.

4.3.1.1 Consistency

To ensure that users share a sense of space and presence within the VE and to enable accurate manipulation and interaction, the environment databases must remain consistent. This consistency needs to be ensured *between the distributed environment servers* and *between the servers and the clients*. Any actions that take place must be propagated to all interested parties, the updates must remain ordered and any conflicts must be resolved or avoided as appropriate to the environment and to the situation in which the conflict has occurred.

A conflict could occur when two users wish to interact with the same object at the same time. In certain situations, these conflicts can be avoided, such as in a virtual shopping mall where users wishing to examine items can each be given a duplicate object to view. In the single server design, this does not pose a problem as the first-come-first-served nature of the networking and the processing of requests ensures that only one request is processed at a time no matter how close together they arrive.

Only one client can therefore have control of the object at any one time (as the other request would be rejected). In a distributed server situation, a request could arrive at two separate servers. Unless the servers negotiate these control requests, the two clients could both be given control of the object and a consistency error will occur when both clients attempt to manipulate the object. A means of negotiating the control of objects within the environment will be discussed later in this chapter.

To maintain the environment database consistency across all the servers, the communication between the servers must be guaranteed. In addition, the same data may need to be sent to interested clients and this communication must be guaranteed. Research has been done into methods to maintain the consistency of environment databases in the face of message delivery failures and message delays [Vog01], although these concepts do not often scale well or rely on systems with significant overheads that introduce delays into the system. Since one of the main goals of the design is to ensure RT responses to client interactions, it is important to ensure that these RT responses are met while still guaranteeing the delivery of the data,

A means of limiting the overall communication delays is to reduce the quantity of data being sent between the servers, and between the clients and the servers. If the servers propagate all the data they receive to their peers, the server cluster would soon become congested and a bottleneck. A means of filtering this data while still maintaining consistency and remaining within the latency bounds is therefore important and this will be discussed shortly.

Another problem associated with data communication and consistency is that of the data ordering. If data arrives out of order and is applied to the database, it will cause inconsistency in the various databases (client and servers). If the data is queued and reordered at the receiver, extra delays will be introduced which may damage the RT responsiveness of the environment. Fortunately, since the design is aimed at using a ATM which is a connection-oriented networking protocol, the problems often associated with data ordering are not present as data cells will always arrive in order even if intermediate cells have been lost. This simplifies data structures as ordering information does not need to be sent with the data and the data overhead that this information would require, is removed.

4.3.1.2 Latency

To ensure realistic interaction with the VE, communication must occur in RT. By RT it is meant that the maximum response time to client interactions with the environment must be no greater than 100ms

with a maximum of 10ms positive[21] jitter [Wlo95] [Str00]. As the end-to-end latency is not easily guaranteed, a framework of maximum delays between environment components needs to be set out to set an upper-bound time framework that must be guaranteed by each individual component in the path of the data.

Since the environment servers are fully meshed, to traverse the server mesh requires a path containing only two servers and one communication link. Communication from an enactor to receiving viewer will therefore occur, at most, between two clients, two servers and over two client-server communications links and over a single inter-server link. This is shown in Figure 4.6 below.

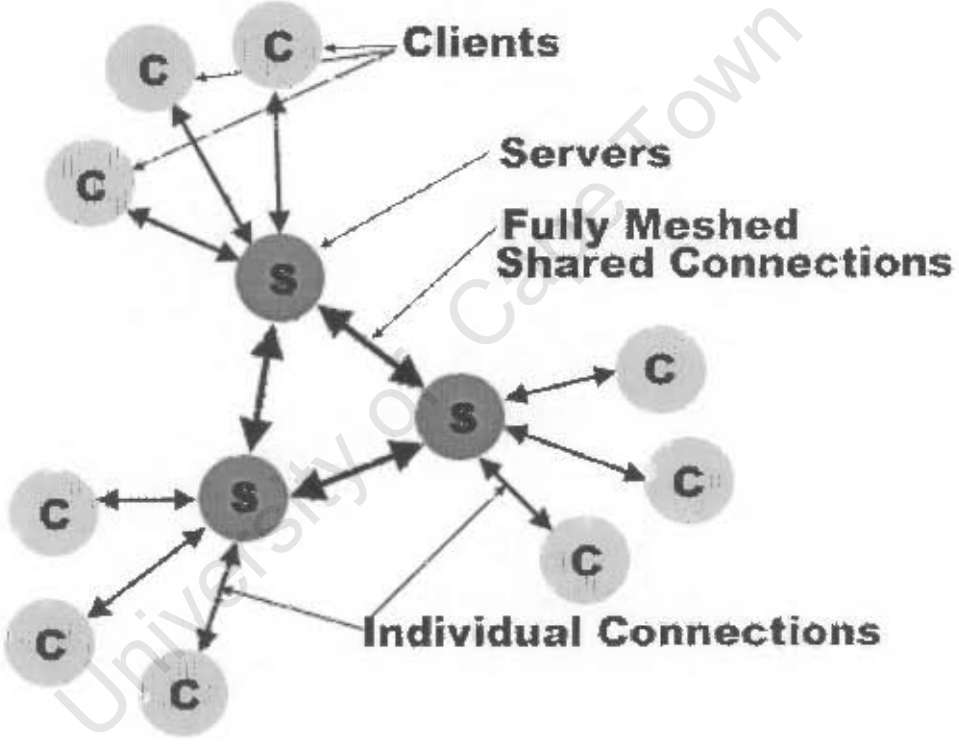


Figure 4.6: Client-Server and Inter-Server Connection Framework

Figure 4.6 above shows how clients connect to a meshed centralised environment server. The mesh extends to larger numbers of servers with each client having a direct connection to every other client which ensures that information sourced at a single client will reach any other client having to travel through at most two servers. Note that the inter-server connections are shared connections and any

[21] The Jitter only needs to be measured in the positive direction as communication that takes less than 100ms is preferred, while communication that takes longer than 110ms will compromise the interactivity of the environment.

data travelling between the servers will have to be multiplexed at the servers before sending. The clients connect to their specific server with a dedicated connection.

The fact that there are a limited number of logical entities between the source and destination for any data allows a logical framework to be constructed that sets upper bounds on the time spent in these various logical nodes. The nodes include both of the servers, the clients and the various communication media. A single source-destination pair is shown in Figure 4.7 below.

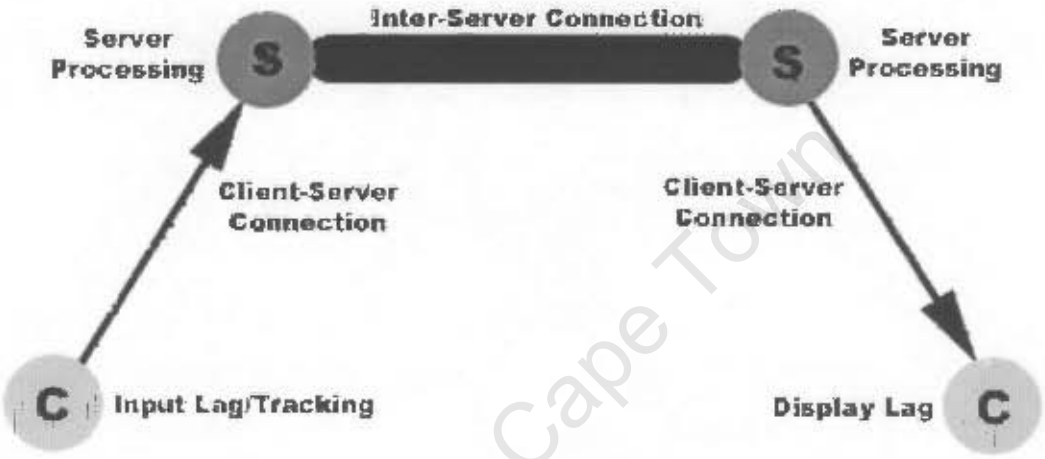


Figure 4.7: A Single Source-Destination Pair

Figure 4.7 shows a single source destination pair, clearly showing how the clients use dedicated links to connect to the servers and how inter-server connections are shared with any other communication that is taking place between the servers. Figure 4.7 also labels the sources of the delays showing that the delays occurring at the source-client result from input lag and tracking, and at the destination client where the data is displayed (this can be to a non-visual output, like sound). Note that the input and display times must be included in the end-to-end limits as the 100ms limit is the time from the user (not the client) executing the action to the receiving user (again, not the client) experiencing the result of the executed action.

Research [Wlo95] has show that 10ms is required by the client for retrieval of data from appropriate input sources, filtering of this data, encapsulating the remaining data into appropriate data packets and finally, sending the data. The same research also shows that 10ms is needed by the client to process received packets of information and rendering the data appropriately. Rendering, in this instance, covers all forms of data output to the user. These limits are physically possible to meet with most modern display hardware and with the various other output devices that can be used. If the hardware is

not able to meet these limits, the complexity of the data being rendered will have to be reduced. Similarly, the 10ms limit can be met by most input devices and if the system is not able to attain the needed information within 10ms, the user will have to be prompted not to use the input device. For the remainder of the design, it will be assumed that the client software is able to ensure these limits.

The remaining 80ms must then be divided across the various logical nodes seen in Figure 4.7. While there is no optimal solution to this problem, it is noted that there are several considerations that need to be taken into account when dividing the time over the various logical components:

- Inter-server communication should have stricter latency bounds to allow user connections to have longer latencies. This will allow users with connections of lower quality to still connect to the environment and experience it, but this will result in the inter-server connections costing more since they require a higher quality of service from the service providers supplying the connections. While this is a drawback, it is noted that the data sent between servers is multiplexed and will result in a somewhat lower cost than if a number of inter-server connections were to be used or if the client-server connections where forced to be of the higher quality.

- Communication, while seemingly instantaneous over short distance, can result in considerable delays. A trans-Atlantic crossing will yield between 25 and 30ms delay due only to speed-of-light propagation delays and the slowdown of electrical signals in the cables [Sme01]. A round trip transatlantic communication had delays of over 120ms in [Kin98] which used IP over ATM to test a long-distance DVE and in August 2001 the delays were measured at 79.955ms [Sme01]. The server time should therefore be kept to an absolute minimum to enable time to be given to communication occurring between the servers, and between the servers and the clients.

- While the client-server and server-client communication times do not need to be symmetrical, it does simplify the design of the overall system and the testing of any implemented software. It is noted that an asymmetrical system may be more suited to the design of a VE as the client will naturally be receiving more data from the multiple client-sources than it will be sending and thus will require more bandwidth on the downlink. While this does not affect the delay limits associated with this data transfer, it could affect the availability of the connections and the increase in delay limit could allow more connections to be made at the required delay values. This topic is not considered further in this dissertation but is one for further research.

- The time delay at either server should be the same as the functions performed at either of the servers are identical. Since the servers are dedicated applications that are more than likely running on dedicated hardware, their response times should naturally be faster and their response times limits can be more stringent.

Having noted these factors, the framework shown in Figure 4.8 was decided upon.

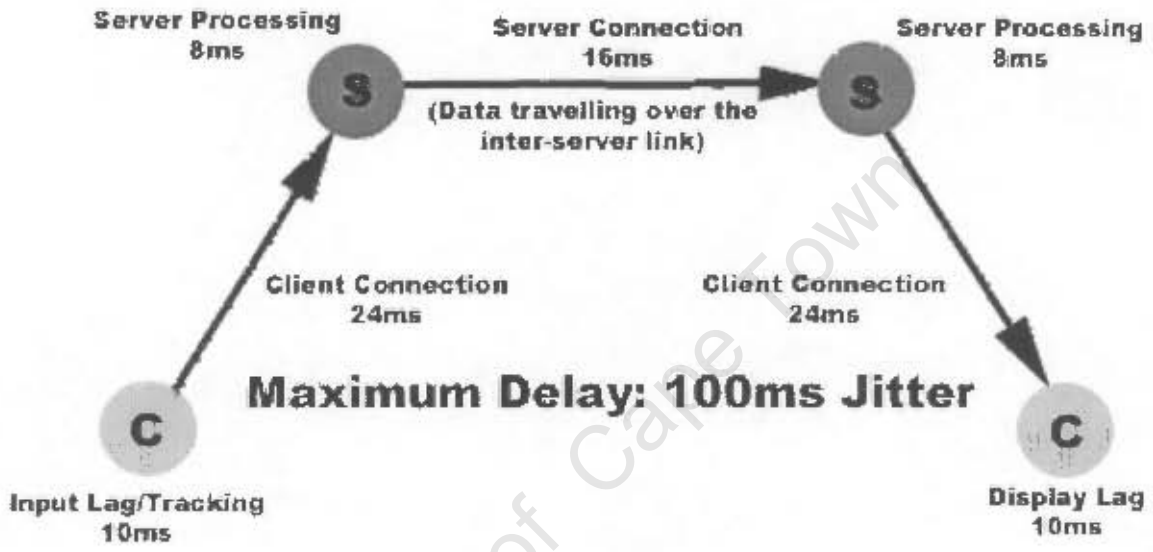


Figure 4.8: The HEAVEn Maximum Delay Framework

Each server has 8ms to process and distribute the data received and their tasks must be completed within this time. This can be considered a *hard* RT deadline if the servers are being implemented on a RT platform and the 8ms is the time measure from the initial receiving of the packet from the application level input communication socket, to the time it is sent from the output socket. Between the servers, there is a 16ms network transmission time, and between the client-server pairs, a 24ms maximum transmission time.

Network Segment / Time in milliseconds	Maximum Latency
Sending Client Interaction Processing	10ms
Propagation Delays	24ms
Sending Server Response Times	8ms
Propagation Delays	16ms

Network Segment / Time in milliseconds	Maximum Latency
Receiving Server Response Times	8ms
Propagation Delays	24ms
Receiving Client Rendering Time	10ms
Total Time	100ms

Table 4.1: The HEAVEn Maximum Delay Framework

Table 4.1 above illustrates how stringent the delay requirements are for communication within a DVE and how the available 100ms is divided over the various logical components within the HEAVEn architecture.

ATM, and other connection-oriented network protocols, is known for relatively lengthy connection establishment times in comparison to other connectionless networking protocols. In ATM, this is due to the connection-oriented nature of the connections that must negotiate a QoS contract with every node in the connection's path. While connection set-up times may factor as a problem in many architectures, they do not factor into the latency of the communication in the above framework. This is because the communication channels between the servers, and between servers and clients, are created when a new server or client *joins* the environment and the connections are not recreated for each packet of data being sent.

There are methods to compensate for data that is late and exceeds the delay limits noted in Table 4.1, although these methods tend to be client specific and smooth any such situations over through the use of movement prediction (such as dead reckoning [Roe97]), object caching and by using proxy-like object imitators [Lea95]. This client-side functionality is also being researched at the University of Cape Town [Doy03+].

4.3.1.3 Limiting the Distributed Data

No matter how computationally powerful the server hardware and how large the network bandwidth used, the data being distributed will eventually grow to a point where a client-server architecture will become congested. If schemes are not put in place to limit the quantity of data that is being sent between the servers in the DVE architecture and between the servers and the clients, then the system will rapidly become congested and the scalability of the design will stunted. There are three ways of limiting the data being sent: The first limits the quantity of data being distributed, the second considers

the tagging of non-critical data being sent for network discarding and the third filters selected received data from the system, resending only an 'appropriate' data set.

An initial approach is to ensure that the data being distributed is small enough to fit into the smallest number of bits possible. It is impossible not to consider the network layer here as the limiting of the data to a single ATM cell would significantly reduce the data being sent and at the same time, reduce overheads associated with any erroneous data as partial data loss could never occur. Ensuring that data is not reliant on other data being sent also ensures that errors are not compounded should data loss or corruption occur.

It can be considered that, while data delivery in the environment cluster and between clients and the environment servers is guaranteed, some of the data is not 'critical' to the environment. If this was the case, this data could be 'tagged' as not requiring any acknowledgment, or at a lower level, tagged so that if the network becomes congested, it could be discarded by the network itself. While this is not easily implemented, it is an option for application aware networks.

In the naïve design, filtering can easily be implemented at the centralised environment server before the data is sent to the clients. Any data that was filtered could still be applied to the environment database. In the distributed version, the filtering becomes somewhat more complex as the other servers often need the information to ensure that they maintain a consistent database. In addition, users may be connected to other servers that require the update data and it can thus not be filtered. The requirements regarding how the data will be filtered will be focused on shortly.

4.3.2 Describing the Environment - Descriptor Files

Before we can continue describing the environment servers, it is important to define how the servers are able to store and define the environment, how the clients can understand the structure of the environment to be displayed and how the environment servers limit the size of the environment database that the clients must download. This task requires what is called a 'scene description language' and there are several available [Pre96] though none is truly useful for what was felt to be necessary in the HEAVEn design. Unlike other scene description languages, the HEAVEn design describes the environment by constructing the scene from hierarchically interlinked *descriptor files*. As the name implies, these text-based files are used to describe the virtual environment. The files

contain data describing the logical space around the user's avatar and the media files used to represent the environment to be rendered. This includes:

- Basic object information (whether that be model, picture or other media data) used to represent simple objects in the logical area. The data includes the position, orientation and other object specific flags as well as a unique identifier representing the file to be downloaded from a MS.

- Any light sources and their intensity, position and type. As users cannot see without light, there must be at least one light source available although this can be described in the root descriptor file and can simply be an ambient light source.

- Links to descriptor files representing subspaces within the logical space around the user's avatar. The data describing the subspace includes the bounding area in which the subspace is active and a unique identifier representing the descriptor file that describes the subspace and the models within it. If the user moves an object into the area stipulated by a subspace's boundaries, the object will be removed from the current descriptor file and added to the subspace's descriptor file.

- A unique identifier that represents the parent descriptor file. The parent descriptor file, or superspace, sees the logical space in which the user's avatar is moving as a subspace of its own area and stipulates its boundaries. The client must retrieve the first parent descriptor file to retrieve the bounding data for the current area as duplication of this data in two files could cause data inconsistencies if the bounding area is altered in one file but not in the other. If an object moves beyond the bounds of the current logical space, that object must be included in the parent's logical space. Similarly, if the user moves beyond the logical space, the client would then have to render the models information and status as described by the parent descriptor file. This allows simple linking of areas in the VE and is a logical tree structure that limits the size of the descriptor files to manageable portions that can be easily retrieved from the environments servers, by the clients, when needed.

An example of how the descriptor file works could be a descriptor file for a dining room inside a palace. It will contain model information for a table, a subspace for the fireplace and an ambient light source. The subspace could itself reference the objects within it, such as the wooden logs in the

fireplace as well as another light source. The file will also include a link to the parent descriptor that links the dinning room to the first floor of the palace. The example descriptor file is shown in Figure 4.9.

Key	Data	Comment
Parent	Rooms\Palance\FirstFloor\firstfloor.desc	// The path of the Parent descriptor file
Model	18458348	// A unique model ID
File Path	Rooms\Palace\FirstfFloor\Dining\table.wrl	// The file reference for the model
Position	0 -50 75	// The position of the object relative to the space
Orientation	0 1 0 0 0 1 0 0 0 1	// The orientation of the object relative to the space
Scale	4 1 1	// The scale allowing objects to be correctly resized
Flags	0	// Any flags that affect the object
Subspace		
Descriptor Path	Rooms\Palace\FirstFloor\Dining\fireplace.desc	// The path of the subspace's descriptor file
Centre Position	0 -50 0	// The centre position relative to this descriptor file
Orientation	1 0 0 0 1 0 0 0 1 0	// The orientation relative to this descriptor file
Bounding Box	20 20 30	// The size of the bounding box for the subspace
Light		// A visible light source (can't see in the dark!)
Type	Ambient	// Could be Ambient, Directed, Point, Spot etc
Position	100 100 100	// The source of the Light

Figure 4.9: An Example Descriptor File for a Palace Dining Room

The hierarchical nature of the descriptor files allow the environment to be broken into simple Levels of Detail (LoD). This enables the client software to choose how much of the current environment it wishes to display and how much of the surrounding environment the client wishes to retrieve and include in the rendering of the environment. In our example, a client rendering the dinning room could choose to display the table but ignore the more complex fireplace, offering simple wire-mesh bounding boxes to show that objects exist in the area of the fireplace. Should the user wish more detail, they can focus on the representation of the fireplace by approaching it or pointing at it within the logical environment. The client could then render it, leaving other details out of the environment to compensate for any client limitations. In addition, while the client is rendering the dinning room, it could at the same time be pre-fetching and caching [Ari99] [Par01] other object models used in the parent superspace (the first floor) and the various linked subspaces (the fireplace).

An important result of breaking the environment database into the smaller descriptor files is that it allows a local referencing system. Objects described within a descriptor file reference the descriptor file's bounding box rather than a global referencing position. Position and orientation values are therefore relatively small compared to similar global references. This enables similarly sized (in terms of bits used to store the values) references to be more accurate and conversely enables fewer data bits

to be sent when updating a position and orientation value. 'Locales' [Pur99] [Pur00] are a concept used to divided an environment into multicast groups in both the HIVE / MASSIVE DVE and has similar advantages though the separation of the environment does not follow the lines of the descriptor files.

Note that the position of the user's avatar is not stored in the descriptor files. This is to ensure that the descriptor files remain as small as possible as clients will regularly retrieve this information and in a large-scale VE, the number of users that could have disconnected in a certain area could make the descriptor file large and the downloading of this file would be wasteful of bandwidth. Descriptor files are requested when users connect to the environment, when users move to different areas within the environment, and when the client is caching information about the environment around the user's avatar. The last position, orientation and other user specific data must be stored separately by the environment servers for retrieval when the client next connects.

In addition to storing links to other descriptor files, links to *portals* that lead to other VEs can be included. These portals allow the client to gain information about other VEs. This information can then be used to disconnect from the current VE and to connect to a new one. Only the name of the environment that the user and client will join is needed thus the inclusion of portal information is as simple as knowing the name of the destination environment. The only client specific requirement is that the user might need to be registered with the new environment, as they may need authentication when connecting.

4.3.3 Separating the Data Streams - Control and Broadcast Streams

The distributed environment servers receive information from the clients regarding the actions and movements the users are initiating within the VE. The servers mediate these actions to ensure that conflicts are prevented before they respond to the originating client and broadcast the data to interested clients. While the tasks of mediation and rebroadcast can be performed together on an environment server, the data that must be resent and the data that is a response to a mediated task have differing requirements. These differences can be exploited if the connection to the client is divided into two, with the individual connection having appropriate functionality associated with it.

The first stream, labelled the control stream, consists of data that must be guaranteed, as losses in the data stream can often be critical to the consistency of the environment as a whole and to the user's

interaction and immersive presence within the environment. This includes database updates, control requests, descriptor requests, object creation notifications and other updates that can be considered "once off" and are critical to the consistency of the environment.

The second stream, labelled the broadcast stream, consists of data that can sustain small losses due to the duplication and temporal nature of the data. This data includes continual update packets referring to user and object positions, voice data and other streamed media that does not benefit from any data guarantees and which is easily replaced by newer data rather than resending the older information.

Both control and broadcast streams must remain RT in nature to ensure that the RT interaction within the environment is ensured. The connections must therefore obey the latency requirements tabulated in Table 4.1.

The functionality of the environment servers is therefore divided into two areas, each producing a stream of data. The control server produces *control* data while the broadcast server produces *broadcast* data. While the data can be easily separated at the client before sending it to the appropriate server, there must still be communication between the broadcast and control servers as will be discussed shortly.

4.3.3.1 The Control Servers (CSs)

Requests for control of objects, the notification of object creation and destruction, the announcement of user connections and departures, the storing of user location information, and the storing, updating and distribution of the descriptor files are the main tasks associated with the *Control* aspect of the logical environment server. These tasks require the guaranteed delivery of the associated data to ensure consistency in the environment. The tasks, discussed further here, illustrate the need for the guarantees.

- The *distribution of descriptor files* must be as any error in the data could result in an inconsistent view of the environment.

- The *mediation* of the object creation, destruction and control of objects must be guaranteed to prevent interaction conflicts between users, as this could cause inconsistencies in the environment databases. This also ensures that the environment descriptor files are correctly updated when the objects are *created*, *destroyed*, *locked* and *released*. The need for mediation

occurs when there are conflicting needs in the environment. An example of this could be two clients wanting to take control of the same object at the same time.

- *Announcements* regarding the creation and destruction of objects, the locking and releasing of objects, and the connecting or departing of clients will only produce single updates that must be guaranteed or they could be lost causing inconsistencies in the environment databases. The disconnecting of a client can occur without the user exiting the environment and it is the control server's task to monitor the connections and announce the unforeseen departure of a client.

- In addition to the three announcement areas noted above, other areas can require guaranteed delivery and control to ensure environment consistency. Movement prediction algorithms often require that the ending location data be guaranteed. These prediction algorithms, such as the dead-reckoning algorithms [Roe97] used by SIMNET and DIS, often limit the update data sent by the clients by predicting the movement of objects. The data being sent is thus far more important to the consistency of the environment and must therefore be guaranteed. The choice of using a prediction algorithm that increases the importance of the data but reduces the quantity of data sent is a trade-off that needs to be further researched. This topic is not developed further in this dissertation. Similarly, the "killing" (not the actual destruction of the object within the environment) of a unit in a military simulation or game needs to be guaranteed and announced to prevent the unit from continuing to function as it would if the update was corrupted. The additional announcements are environment specific but have similar functional needs.

When clients leave the environment, the control servers store data regarding the avatar's last location. In the case where the client does not intentionally disconnect, the control server stores the last known position (note that the last known position might not actually be the last position of the avatar, as movement via prediction algorithms is not taken into account). This information can then be retrieved by the client when they next connect to the environment. It is stored separately from the information stored in the descriptor files to ensure that they do not become cluttered with data about users who are not logged on.

When a client joins the VE, it connects to a control server to retrieve the last stored position of the client's avatar, the descriptor file of the virtual space in which the avatar is connecting, and a list of

users connected in the same virtual space and their locations. An announcement message is then sent to all interested clients to notify them of the connection of the new client.

Several VE architectures, such as the well known SIMNET and DIS, rely on the connected clients to update a new client with appropriate data regarding their current position. This method of updating the new client cannot be guaranteed as users could "hide" by not announcing themselves and this prevents true environment consistency. Similarly, the control aspects of the environment cannot be left to the client to perform as failure to complete any of the control functions will result in damage to the consistency of the environment. The interaction aspects can, however, be left to the client as the client must send interaction information to the environment otherwise the client cannot interact with the environment. If these guidelines are adhered to, there should never be a situation where a client could jeopardise the consistency of an environment by failing to send information.

The transport of the control data must be guaranteed and the data must be transmitted in RT to ensure that the environment remains immersive. A positive acknowledgement protocol is therefore used to acknowledge control data received. Note that the sending of acknowledgements data does not affect the transfer times for successful data delivery although failed data delivery will result in extra delays as the data will have to be resent. These delays will likely affect the overall transfer delay, but it is imperative that all the data be delivered and delivered correctly.

While the communication between the clients and control servers has been focused on, there must also be communication between control servers as the control updates must be sent to all the servers to ensure database consistency and in addition to this, clients on other servers will often be interested in this information. The inter-server communication must also be guaranteed and a positive acknowledgement protocol can be used between the servers to ensure the delivery of the data.

To reduce the network overheads associated with the distribution of the data to the control servers, the servers must be fully meshed as described earlier. Since every control server requires all the control data produced, this control server mesh would be an ideal broadcast or multicast service candidate if it was not for the requirement that the delivery of the data be guaranteed. As such, multiple point-to-point connections will need to be used.

As mentioned earlier, certain interactions within the VE require the control servers to negotiate the outcome of these actions as they could conflict with other actions initiated on other control servers at

the same time. These negotiations cannot be as simple as using a first-come-first-served approach as the distributed nature of the control servers makes finding 'who came first' a problem and without timed synchronisation between the servers, this is impossible. The solution chosen for the HEAVEn architecture is to distribute the possibly conflicting control update along with a randomly generated number. If two conflicting updates are received, any control server simply discards the update with the lower random number and distributes the higher on. The sending server must then wait the inter-server transmission time to ensure that it does not receive conflicting updates. If it does not, it will send the requesting client a positive reply, knowing that the other servers have already sent updates to their clients, or a negative reply if it received an update with a higher random number, updating all connected clients with the appropriate update. This solution increases the data overhead for the control communication, as a random number would have to be included in all conflicting control data, but offers a rapid means of solving conflict situations as no data needs to be resent. This trade-off between increased overheads and reduced latency leans towards the issue of latency as negotiation processes can lead to significant delays. When considering the time taken by this negotiation process, it is noted that it does not break the latency limits as the comparison of the random numbers yields an immediate response. Should a more conventional negotiation process have been used, the delay would exceed the 100ms response time limitation.

As noted earlier, the CSs keep track of the position of objects and users within the environment. This is not always possible as moving objects do not necessarily send *control* data. This is especially true when movement prediction algorithms are used to reduce the number of movement updates that are being sent to the environment servers. The choice of continually receiving update information about the movement of objects within the environment might seem like a reasonable means of ensuring that the CSs are able to ensure that the descriptor files remain accurate, but this increases the overheads associated with processing this data and this would rapidly result in congestion at the CSs. As such, the CSs must rely on regular control requests and updates from the clients to update the positions of the objects within the environment. This is a reasonable assumption to make when considering how and when control updates are generated. In addition, if the broadcast data is distributed to the control servers at a very low rate, with older data being replaced with newer data as it is being collated, the CS can regularly receive a 'snapshot' of current state of the environment. This reduces any 'long-term' inconsistencies that may occur should a user simply continue moving in a 'predictable' fashion within the environment (no major prediction algorithm changes to send via the control servers), or in a small environment area (no descriptor file requests) thus producing no control updates.

A client retrieving descriptor files would therefore get a 'near to accurate' descriptor file and would have to rely on the updates received from active clients to correct small errors that could occur in the environment until the client is up-to-date. Client-side smoothing algorithms can be used to remove any obvious inconsistencies so that they do not affect the immersive experience of the user.

As the CSs maintain a regularly updated view of the entire environment, they can calculate the area around a specific avatar in which the user is *interested.* This 'area' is often used for various filtering algorithms that can be used to filter both the control and broadcast data. This will be discussed shortly.

4.3.3.2 The Broadcast Servers (BSs)

The efficient distribution of update data to a selection of users within the VE is the main task associated with the *Broadcast* aspect of the logical environment server. This task requires that the delivery of the data:

- Is completed within the end-to-end delay limits. This ensures that the latency requirements for the user's interactions are met.

- Has the absolute minimum data and processing overhead possible. Most user updates will be sent via the BSs and any savings in these two areas amount to huge savings when multiplied by the number of data packets that will be sent.

- Need not be guaranteed. Updates sent via the BSs contain information that is regularly being updated and small losses can therefore be tolerated due to the natural redundancies in the data being sent and received.

- Be limited by the users' needs. Each user specifies a rate at which they can receive information and this rate is ensured via the use of a *leaky-bucket* (or other similar algorithm) where data is queued before sending. Data is collated in this queue before sending, thus newer data arriving about a specific object, replaces older data

- Not be filtered through any means where the data is interrogated, as this significantly increases the processing overheads on the BSs. Instead, a list can be supplied to the BSs relating input sources to output queues. This list simply filters where the data will be distributed and must be

calculated externally. This enables a BS to *switch* the data from an input network socket to a destination output queue (where it will be leaky-bucket queued as noted above). This simplifies any implementation to a set of pointer operations and thus significantly reduces the processing overheads.

The separation of the broadcast operation from the control and calculation of the filtering lists in the logical environment server, allows the BSs to be dedicated data distribution nodes. This ensures that the BSs are able to handle larger numbers of connected users and this further distributes the load of the logical environment server within the environment cluster.

As noted above, the data that will be distributed by a BS tends to be temporal in nature and consists of such update information as the movement and position updates to objects and users within the environment. The fact that the updates occur regularly as the object moves and the clients process their user's interactions ensures that if some data is lost, it is only a transition between updates that is lost and this transition can be *smoothed* by simple client-side rendering algorithms.

To ensure that data loss is non-critical, it is important that the data being distributed is independent of other data sent. As an example, a scheme that updates a user's position relative to their last known position would need guaranteed delivery of the data as loss of a single packet could have serious consistency repercussions, especially if the initial update was lost. If lost data is independent of the other data being sent, the client can usually smooth over these inconsistencies.

Relevant data should also be as small as possible to enable it to fit into the smallest number of ATM cells possible. Ideally, the data should occupy only one ATM cell of 48bytes. Therefore, if a cell is lost, only a single piece of data is lost and the environment consistency can be easily recovered when the next update is received. Similarly, if a single cell is lost, the other cells that would have been part of the update packet do no congest the network unnecessarily.

While this solution does offer several advantages, it does limit the design of the client in that the standard *difference* updating methods cannot be used. Fixed reference values must therefore be used and while this might not seem a problem, it increases the quantity of data that must be sent with every update. This is not a significant problem as the increased accuracy and reliability gained from using fixed reference updating ensures that the environment remains far more consistent and enables the use of non-guaranteed broadcasts. In addition, as the entire environment is segmented by the descriptor

files, the values in the position updates will remain small as the values are relative to the surrounding bounding box and not to a globally defined reference point.

When considering that not all interested clients are connecting to the same BS, the BSs must be interconnected to pass on information for distribution to the interested clients. These interconnections must also be RT in nature and while the CSs must ensure data distribution to all other CSs, the data sent by a BS only needs to be sent to BSs with interested clients, and to a CS for the 'snapshot' update. While the connections between the BSs are not always used, they must be maintained, as the creation of connections when they would be needed would introduce extra latency, as the connections are re-established. A fully connected mesh of *point-to-point* connection should therefore be maintained between all the BSs. Note that, while multicast connections between the BSs might have seemed ideal because the BSs do not require guaranteed delivery and that the use of multicast connections would have significantly reduced the bandwidth used between BSs, it is important to realise that the BSs pass data on to those servers require it and those servers only.

4.3.3.3 Limiting the Distributed Data

Both the sections defining the BSs and CSs state that data is distributed to clients that are *interested* in receiving it. In a small-scale VE, this could mean that all the users receive the information, but in a large-scale DVE, this is clearly not the case. Given the real world example of two people in different parts of the world, who are not communicating in any way, why would they be interested in each other's actions? The answer is simple, the information is extraneous and simply adds to both the client and server side congestion and it must clearly be filtered. The architecture must therefore include ways of limiting the data being sent and received by both the clients and the servers, and the HEAVEn architecture includes two possible methods. The two methods are *per connection data collation* or filtering, and source grouping or *interest management*.

Per connection data collation is based on the concept that every client in the HEAVEn architecture is able to stipulate their needs and capabilities in terms of receiving information. These needs are limited by their processing power, their physical connection specification and their rendering ability. These limitations can be used to further specify a QoS specification for the connection between the client and the server and will be discussed in the Section 4.4 where the Networking is focused on. Given that each client is therefore able to stipulate its capabilities, the server can tailor the quantity of data it sends to an individual client. To ensure that the servers meet these requirements, every connection to a client is prefixed with a leaky bucket FIFO queue to which new data is added. As data in the DVE is often

temporal in nature, the data in the queue ages rapidly if it is not sent (Control data does not age as rapidly due to its critical nature). If new data arrives, the older data is no longer useful and can therefore be replaced in the queue with the newer data. Note that the new data replaces the older data in the queue, ensuring that the response times for the original data are still maintained. In this method of filtering, the data is collated based entirely on the sources of the data and it does not in any way interrogate the data to be sent. This is important, as calculations needed to filter data based on its content are often processor intensive and are avoided using this method of filtering.

Instead of filtering dependant on the temporal nature of the data arriving from the various data sources, interest management limits the data dependant upon which sources the client wishes to receive data from. By using such techniques as the *Area of Interest* [Oh97] [Roe99] (Also know as Domain of Interest) [Mor97], *Field of View* [Roe97], *Distance from the User* [Roe97], *Occlusion* [Roe97], *Importance of Presence* [Fuj98], and *N Closest to the User* [Gre99b] algorithms, a set of data sources can be selected for a specific client. These client lists are combined to form an *interest matrix* that can then be used by the CSs and BSs to enable them to distribute their data appropriately. The interest matrix stores the unique identifiers of the various data sources, paired with a list of unique destination identifiers and the BS to which they are connected. The need for the BS information is ensure that the data is distributed to the appropriate BS and to limit server congestion where it is unnecessary. If a BS receives information, it is able to look up the destination and simply switch the data to the appropriate output queue while a CS acts similarly although it will always send control updates to all other CSs. This significantly reduces the overheads associated with redistribution of the data within a DVE.

To calculate the *interest matrix* for a specific user, the calculating process needs to have information about:

- The position of all the objects in the environment.

- The user specified parameters that dictate the user's needs. This includes the user's ability to display the environment and receive data. This information will need to be sent directly to the CS that a client is connected to and can be dynamically altered if the client feels it is incapable of receiving the quantity of information it is receiving or if it feels that it can receive more detailed information from a larger area of interest.

- The user's focus within the VE.

- Any server and environment specific parameters or limitations.

As the position, focus and orientation of the user's avatar and the location of the other objects and users within the environment are already available to the CSs, they are ideal servers to host this process. Unfortunately, this introduces extra overheads that will reduce the CSs' performance. If a separate server is created to handle these calculations, the data will have to be gathered before the calculation and this introduces extra bandwidth overheads in the environment cluster. Similarly, the accuracy of the interest matrix is dependant on how regularly the calculations are made and in a fast moving environment, these calculations will need to be updated regularly. There are several obvious trade-offs here that are clearly environment specific. The choice of how often these calculations should be made is left for further research and will more than likely remain an environment specific parameter that would have to be adjusted dependant not only on the type of the environment but also on how the users interact within it. For simplicity, it is assumed that the CSs are able to support the processing needed to calculate the interest matrices and to delivery these matrices to the BSs. The connections used to deliver the interest matrices need to be guaranteed although the data is not time critical.

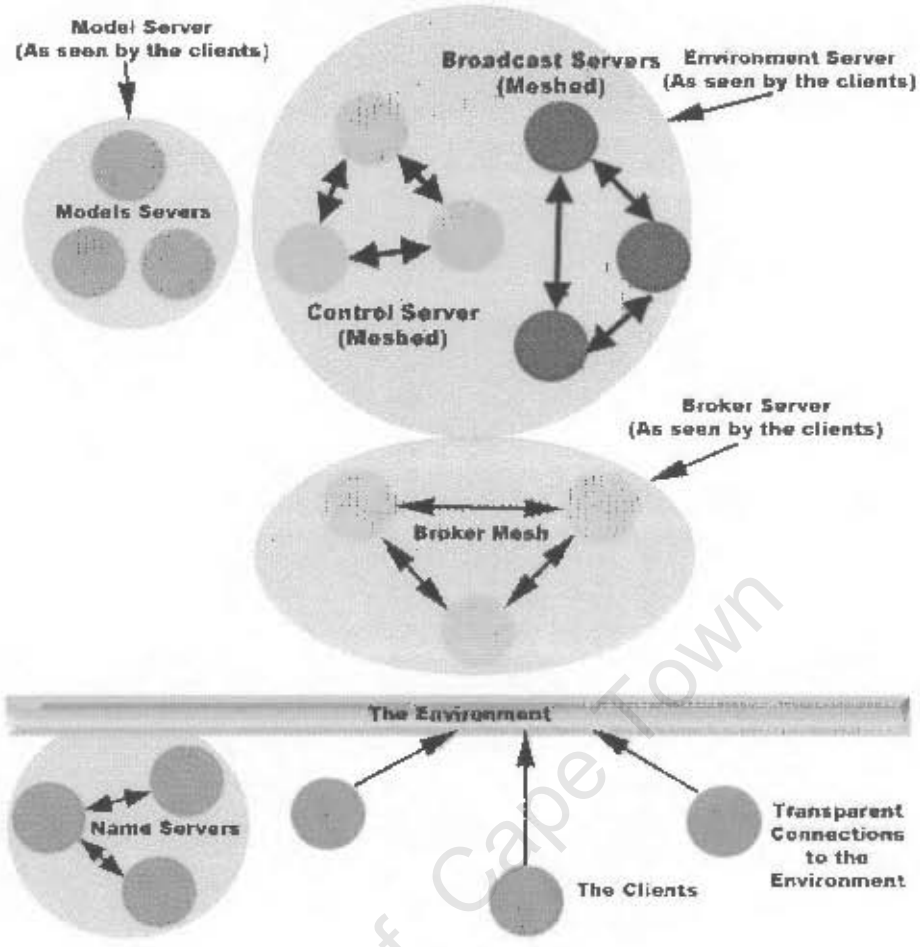


Figure 4.10: The final HEAVEn architecture

Figure 4.10 illustrates the final design showing the Control Server and Broadcast Server meshes and the standalone Model and Media Servers, all forming part of the DVE to which the clients can transparently connect via the Broker Servers that together form a specific VE. The clients locate the environment using the Name Servers that direct them to an appropriate Broker Server, which in term authorises the client and passes them on to the appropriate Control, Broadcast and Model and Media servers.

4.4 The Network Layer

Now that a final DVE architecture has been constructed in the logical application and session layer, the network layer needs to be focused on. This section focuses on how the design's requirements are applied specifically to the ATM network layer that this Architecture has been designed for and how the network is able to support the various connection requirements through the use of the service supplied by the ATM network.

When considering the design, it is important to realise that all of the connections are temporal in nature. This prohibits the creation of Permanent Virtual Circuits (PVCs) for use between the clients and the servers, and between the servers. While inter-server communication may last a considerable time, it should be possible to dynamically add or remove servers without affecting the environment and this would not be the case if PVCs were used [Cis99]. Similarly, PVCs restrict the connections between two servers to a single path. Should a portion of the network in the path of the PVC fail, there will be no way to create the connection. An SVC would, however, be able to find an alternate route if one was available. Similarly, the PVC has to have a set QoS and with the changing demands on the DVE, the connection's QoS may need to be recreated or renegotiated and this is not possible using an PVC. All the connections in the HEAVEn design are thus Switched Virtual Circuits (SVCs). The SVCs used in the design fall into three categories that will now be described in detail.

4.4.1 Guaranteed Delivery and Time-Non-Critical Connections

The first connection type is used for data requiring guaranteed delivery and over which a lightweight positive acknowledgement protocol is used to ensure that delivered data has arrived and is without error. If data has not been acknowledged with a certain time, the data is resent. Note that these connections do not have minimum time requirements and are used for sending time-non-critical information between the clients and servers, and between the servers in the environment. These connections can be found between the:

- Servers in the Name Server Mesh to distribute changes to the Name database, used by any application connecting to a Name Server for authentication or address resolution, and used by the Name Servers to request updates (In this case, from the Broker Servers although the Name Server is non-environment specific and can request updates from any named source).

- Servers in the Broker Server Mesh to distribute changes to the various Broker databases, used by any application connecting to a Broker Server and used by the Broker Servers to request updates from the various Environment Servers (Model and Media, Control and Broadcast).

- Clients and the Model and Media Servers when requests for downloading environment data are made, and between the Model and Media Servers when distributing environment data between these servers when a Broker Server attempts to redistribute file load within the environment. These requests for data sharing also use these connections.

- Between Control and Broadcast servers when the Control Servers distribute Interest Matrices to the various Broadcast Servers.

To summarise, these connections all have three factors in common: The first is that they all require guaranteed delivery. The second is that they are non-time critical, and the third is that these connections can tolerate data loss, as lost data will not be acknowledged and will result in the source resending the data.

When considering the available services offered by an ATM network, the most appropriate choice of service category would be that of an Unspecified Bit Rate (UBR) connection. The UBR connections allow communication over an unspecified communication link and offer no service guarantees to the data beyond a best effort service using any available bandwidth in the network. This is ideal as the communication that will be taking place over these connections does not require time or data guarantees and while it ideally does require that the data eventually be delivered, the positive acknowledgement protocol associated with these connection will ensure that lost data is resent.

The choice of using the ATM Adaptation Layer 5 (AAL5) as the convergence layer will provide the connections with extra payload error correction and detection facilities (32 Bits per PDU) that check delivered data for bit errors, lost cells or cell ordering errors [Cis99]. Data that is damaged is automatically discarded and the lost data will be resent when no acknowledgement is received.

The choice of using the ABR service category was considered since ABR connections can have a Minimum Cell Rate (MCR) specification. This would ensure that the connections would not be throttled. This choice was decided against for two main reasons:

- The newer ATM specifications [Ahm02] consider UBR connections when calculating the overall usage of the data pipe before allowing new connections. This prevents the flooding of the network with UBR connections and at the same time, prevents the throttling of the UBR connections.

- Looking forward to the implementation of the HEAVEn design, ABR connections are not supported by either Microsoft's Winsock2 or by the FORE/Marconi LE-155 and PCA-200EPC [Mar02] [For98] and as such, would be impossible to implement.

When focusing on the connections, it is important to realise that there can currently be no concrete bandwidth usage predictions. The main reason is that many of these connections are on-the-fly connections that are used for a specific purpose such as the request of a Load Index, taking several cells at most, or the retrieval of Model and Media information, the data files for which can become quite large. Where connections are relatively persistent, such as the connections between Broker Servers, the connections will only be used when updates need to be sent to peer-Brokers, which will generally not occur very regularly and do not use much bandwidth.

4.4.2 Guaranteed Delivery and Time-Critical Connections

The second connection type focuses on those connections requiring guaranteed data delivery within a set time. This enables the environment to maintain a realistic interaction with the users and must ensure that data is delivered within 100ms as detailed in the design. These same connections must also guarantee that all the data is delivered, as lost data will damage the consistency of the environment. An application level positive acknowledgement protocol is used to ensure the delivery of the sent data and any data that is not acknowledged must be resent. Note that only the first sending of the data will needs to fall with the 100ms response time, but it is still imperative that the data be delivered. These connections can be found between the:

- Servers in the Control Server mesh to ensure that the updates passing between the control servers arrive thus ensuring the consistency of the environment database.

- Control Servers and their Clients to ensure that interested clients are kept up to data with changes to the environment around them and to do this within the response time limits ensuring a responsive, real-time interactive experience for the user that remains consistent across all the users.

To summarise, these connections require guaranteed delivery and a minimum transfer delay. In addition, while data loss can be recovered using the acknowledgement protocol, data loss should be avoided as it introduces unwanted delays when data is resent.

When considering the available services offered by the ATM network, the most appropriate choice of service category would be that of a Real-Time Variable Bit Rate (rt-VBR) connection. Constant Bit Rate (CBR) is the only other service category that offers RT data transfer though the use of this service category is wasteful of bandwidth as the data sent over these connections is entirely dependant on the activity of the environment and is clearly not constant in nature.

The connection should use AAL5 as the convergence layer as it will provide the connections with extra payload error correction and detection facilities (32 Bits per PDU) that check delivered data for bit errors, lost cells or cell ordering errors [Cis99]. Data that is damaged is automatically discarded and the lost data will be resent when no acknowledgement is received. If data is discarded, the sender will resend the data after a set time (dependant on the sender). Note that the sender might not be the client who sent the data, but rather the sender in a sender-receiver pair along the pathway of the data. This should limit the time wasted waiting for an acknowledgement as the maximum time that a source needs to wait is twice the maximum time limit for the connection added to the maximum time that can be taken by the receiver.

Since rt-VBR connections will be used, the various rt-VBR QoS parameters need to be considered. Other than the timed requirements for the delivery of the rt-VBR data, already set out for the connections in Figure 4.8, the data being sent across the connections must conform to a set of bandwidth limiting traffic specification. So far, the 'bandwidth' used by a connection has not been discussed and this is due to the very random nature of the bandwidth that could be used!

Consider a client connected to a server that can, at worst case, produce a single control update once every 20ms. This will result in the server receiving 50 control updates a second from the client. The control update is a single cell in size. This will result in the server producing 50 acknowledgements a second, assuming there are no errors on the connection. At the same time, the client has an Interest Matrix telling the CS that it is interested in 10 other clients that themselves could, at worst case, produce 50 updates a second resulting in 500 updates being sent to the client every second and the client responding with 500 acknowledgements for a total of 550 updates a second upstream and 550 updates a second downstream. Since an update in this example is a single cell, the worst case Peak Cell Rate (PCR) is 550 cells a second, while the Maximum Burst Size (MBS) and Sustainable Cell Rate (SCR) are both infinite with the traffic actually CBR in nature! On the other hand, the clients could all be standing still and talking, resulting in no control updates at all and thus no control data

being sent. The bandwidth can vary between these two extremes at a moments notice. In addition, as the user moves their avatar through the environment, their Interest Matrix will change dependant on the number of sources of interest around them and while an upper bound can be set, there are no guarantees that the interest matrix will be a set size and thus no guaranteed SCR or MBS.

While the client-server connections have a limited number of information sources (the size of the interest matrix and the client itself), the calculation of an inter-server connection's traffic parameters is far more difficult as communication taking place between the servers is over a common channel and thus all communication is multiplexed. A worst-case scenario can once again be considered, but the requesting of the worst-case traffic parameters for a connection is wasteful of bandwidth and expensive when considering a commercially viable system.

Two options can be used to solve this problem. The first revolves around ATM's ability to offer traffic contract renegotiation should the connection require differing traffic characteristics. A connection could thus begin its life with minimal characteristic requirements and as the client or server requires greater bandwidth, the connections can be renegotiated. This option ensures that environment growth can be handled although it can result in the situation where a connection must grow to ensure the continued environment consistency, but the network is not able to support the new connection.

The second option would work in conjunction with the first and has the brokers storing a set of 'appropriate' values for the various rt-VBR traffic characteristics. When a client or server connects to the broker mesh, they can request these values and thus use them when creating their rt-VBR connections to the environment. The values stored by the brokers will be set values that will, no doubt, have been empirically attained and specific to the environment and its needs. This will ensure that the initial connection characteristics can handle the normal traffic for the environment and will thus not generally have to be renegotiated.

While this does not solve the problem of stipulating a set bandwidth requirement for the rt-VBR traffic, it does offer a situation where the connections can be created with appropriate initial connection characteristics. It also poses a significant area for future research and development, and prompts consideration into how a set of initial traffic parameters may be calculated for use in the implementation of the design.

Another side note that again, does not solve our problem, but does in a round-a-bout way justify the focus of the design on the timed requirement of the connection instead of the bandwidth requirements is that it is a well known fact that, while the capacity of fibre optic connections doubles every 12 months - with the current benchmark at Lucent Technologies being an incredible 34 Terabits per second [Mac02] - the delays remain limited by the speed of light which travels across a time zone in approximately 8.25ms.

4.4.3 Non-Guaranteed Time-Critical Connections

The third connection type focuses on those connections that require only a data transfer time guarantee. This guarantee enables the environment to maintain a realistic interaction with the users. The delivery of the data must occur within 100ms as detailed in the design. While the connections do require time critical delivery guarantees, due to the sent data's temporal nature and the ability of the clients to easily recover from small losses of data, these connections do not need to guarantee the delivery of the data being sent. These connections can be found between the:

- Servers in the Broadcast Server mesh, where data is received and then passed on to interested servers and clients.

- Broadcast Servers and their connected Clients who are interested in the data.

To summarise, these connections require a minimum transfer delay for the data being delivered. The connections must therefore use the rt-VBR service category. The choice of using the CBR service category, the only other RT service category, had to be discarded, as it would be wasteful of bandwidth. Data will only be sent over these connections when the source (be it a server or client) has information that is of interest to the destination (again, server or client) and this is in no way a characteristic of CBR traffic.

Another important factor to consider when deciding how these connections should be chosen, is that the data being sent across these connections is relatively small. In fact, the 48 bytes available in a standard ATM cell is enough to store an entire update packet. If, for example, the AAL5 convergence layer was used, at least 4 of the 48 bytes would be used for packet CRC checks. The extra header information would be added to every PDU sent since the update PDU are relatively small, the per PDU overhead associated an AAL5 connection would be very high. Since there are no guarantee

requirements for the connection, if AAL0 were to be used, there would be no additional convergence overheads and this would also ensure that the full 48 byte payload is available for the update. This in itself is a huge saving in terms of bandwidth though it does require that the update PDU be designed to fit into 48 bytes, which is entirely possible.

Since rt-VBR connections will be used, the various rt-VBR traffic parameters need to be considered and as noted earlier, the connection parameters cannot be easily calculated. The use of renegotiable traffic contracts and the brokers to store 'normal' connection parameters will ensure that the connections are created with appropriate traffic parameters and QoS.

4.5 Possible Scenarios

This section considers several scenarios that could occur within the DVE and traces the resultant data flow through the HEAVEn architecture. These scenarios are here as examples to graphically depict and draw together the design that has been described in this section. The scenarios are all assumed to use security callbacks to the broker server for security token validation. Some environments may simply choose to ignore this step.

4.5.1 A Client Connects to the Environment

The first scenario, shown in Figure 4.11, shows the messages passed when a client attempts to connect to a specific DVE. The messages are labelled and described beneath.

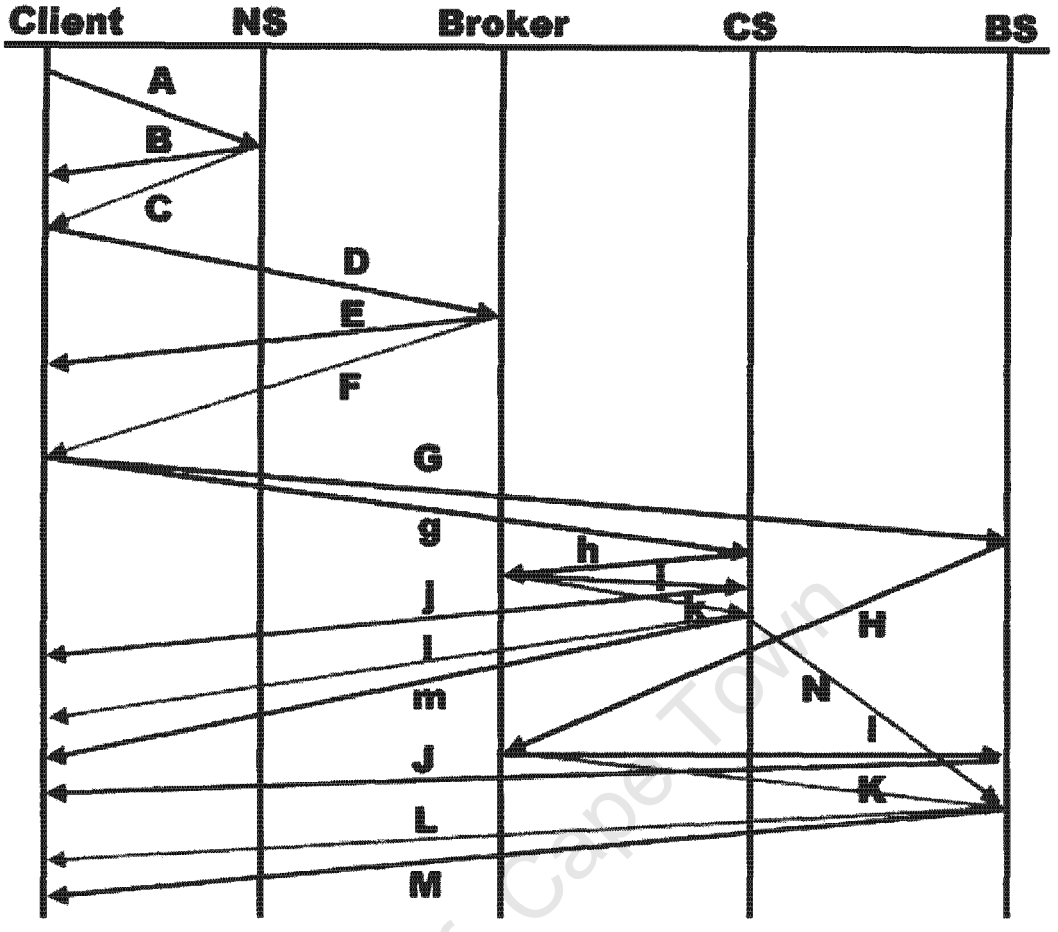


Figure 4.11: A Client connects to an Environment

Key: The *Black* arcs are *Action* arcs

The *Red* arcs are *Negative Responses*

The *Green* arcs are *Positive Responses*

The *Blue* arc is a *Reaction*

The arrowheads direct the flow of data along the arcs between environment entities.

Arc A: The Client requests the address of an environment giving a known Name Server the *Descriptive Name* of the environment server.

Arcs B and C: The Name Server responds to the Client with a negative acknowledgment, shown in arc B stating that the descriptive name is invalid, or a positive acknowledgement, shown as arc C, where the NS returns a list of Broker Servers addresses for the named environment.

Arc D: The Client then steps through the list of Brokers until it makes a connection. When a connection is made, the Client sends its User ID and Password along with a request to connect to the environment.

Arcs E and F: The Broker validates the User's ID and if it is invalid, returns a negative, shown in arc E, or a positive, shown in arc F, along with a security token and a list of environment servers (MSs, BSs and CSs).

Arcs G and g: The Client will then simultaneously step through the list of Broadcast and Control servers, attempting to connect to a server of both types. When a connection is made, the Client will request to join the server and transmit its security token, user ID and any connection specific requirements.

Arcs H and h: Having connected to the server (BS or CS), it connects back to the Broker to validate the Client's security token and based on the response, arcs I and i being negative responses from the Broker, the server will either reply with a negative response to the client, shown in arcs J and j, or it will attempt to accept the connections. If the connections are acceptable, considering the server's load and the connection requirements requested by the client, then the server responds with a positive, shown in arcs L and l, or with a negative, shown in arcs M and m.

When the Control Server responds with a positive, as shown in arc l, it will also respond with the users last known position in the environment (if it is known), a list of the currently connected user in the area close to the user, the descriptor file in which the user is standing and that descriptor file's parent descriptor file. From this, the Client can begin constructing the world around the user's avatar and begin downloading any new model and media data that is not currently cached. The Control Server will also, using the data supplied by the client on connection, begin constructing the Interest Matrix for the Client. Once this is done, it will deliver the Interest Matrix to the Broadcast Servers, shown in arc N. Without a defined Interest Matrix, the Client will not receive any update information from the Broadcast Servers, as they will not be interested in anything! The Control Server will then announce, to all interested clients, that the new Client has connected.

The Broadcast Server's response is a simple acknowledgement and the server will begin sending data to the client as soon as new data arrives at the Broadcast Server to be sent.

These actions completed, the Client is now able to receive update information from both a Control and Broadcast servers and is a part of the environment.

4.5.2 A Client Requests Model and Media Data

The second scenario, shown in Figure 4.12, shows a Client requesting model and media data from a Model and Media server. The messages are labelled and described beneath.

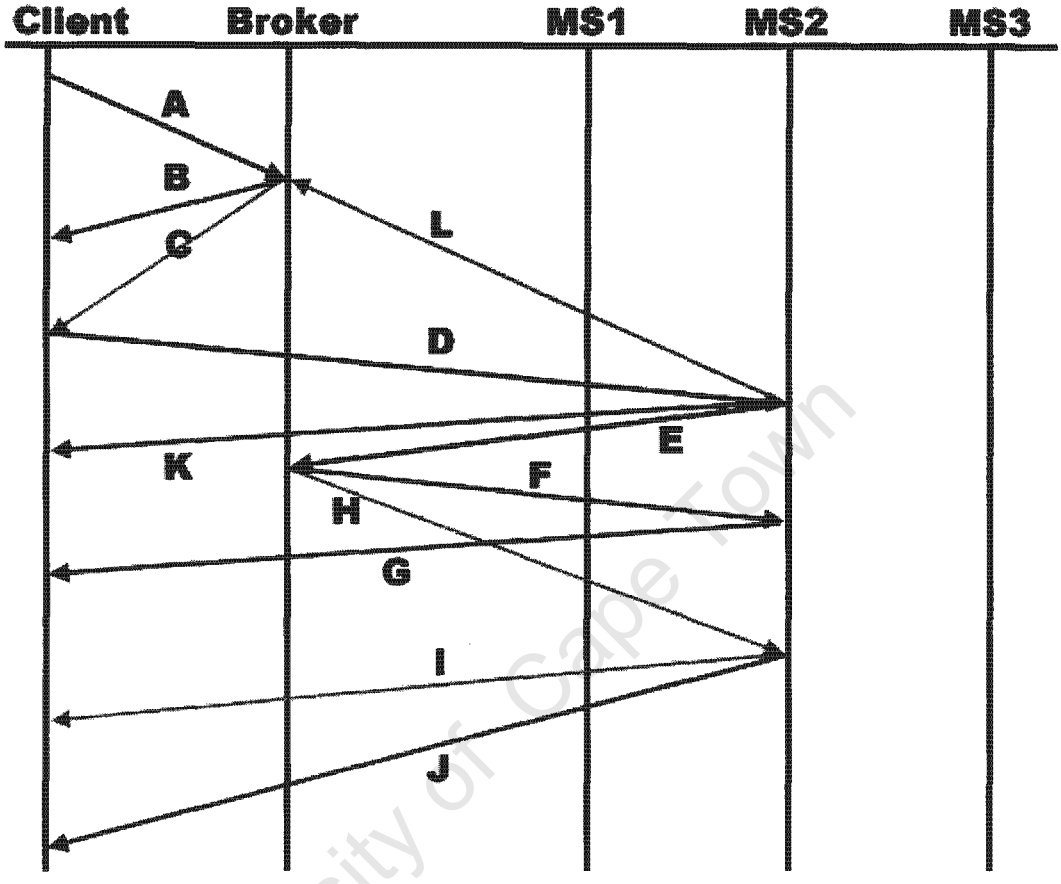


Figure 4.12: A Client Requests Model and Media Data

Key: The *Black* arcs are *Action* arcs

The *Red* arcs are *Negative Responses*

The *Green* arcs are *Positive Responses*

The *Blue* arc is a *Reaction*

The arrowheads direct the flow of data along the arcs between environment entities.

Arc A: The Client, having already located an active Broker, reconnects to it. It then sends its User ID, Security Token, and a list of the unique identifiers for the Model and Media files that it requires. The unique identifiers for the model and media files are stored in the descriptor files distributed to the client when entering or moving within the environment.

Arcs B and C: The Broker validates the user's security token, returning a negative if it is

invalid. If valid, it checks its database of model and media files for the requested unique file identifiers returning negatives for files that are not locatable, and returning the addresses of Model and Media Servers for the files that are available for retrieval.

Looped Action - Arcs D to J: The Client then steps through the list of Model and Media Server addresses and unique identifiers, executing the following set of arcs for each file that needs to be retrieved.

Arc D: Selecting a unique identifier - Model and Media Server address pair, the Client connects to the MS sending the request for the file giving its unique identifier, the Client's User ID and security token. If No connection occurs, the Client will obviously have to contact the Broker again though this should not happen as the Brokers keep up-to-date activity lists of the various servers.

Arcs E, F and G: The MS checks the Client's security token sending the User ID and Security Token to the Broker. If the token is invalid, the Broker responds with a negative, arc F, and the MS returns a negative to the Client, arc G.

Arcs H, I and J: Given a positive Broker response, arc H, the MS must then locate the requested file given the Unique Identifier. It does this by locating the file in a model and media file database thus allowing the storage of the files in MS-specific locations. If the file is not available, the response is negative, arc J, and if the file is available, the file begins transmitting, arc I.

Arcs K and L: If the MS is too loaded to respond to the Client's initial request, it will respond with an immediate negative, not even checking the client's token. It will then respond by updating the Broker directly with its condition to ensure that no other clients attempt to connect to it until the load has been sufficiently reduced to handle more clients.

These actions completed, the Client will have now retrieved the necessary model and media files to display the environment appropriately to the user.

4.5.3 Data arrives at a Broadcast Server and at a Control Server

The third scenario, shown in Figure 4.13, shows both a Control and a Broadcast Server receiving Data, which is to be sent on. Note the differences in how these two entities handle the redistribution of the data. The messages are labelled and described beneath.

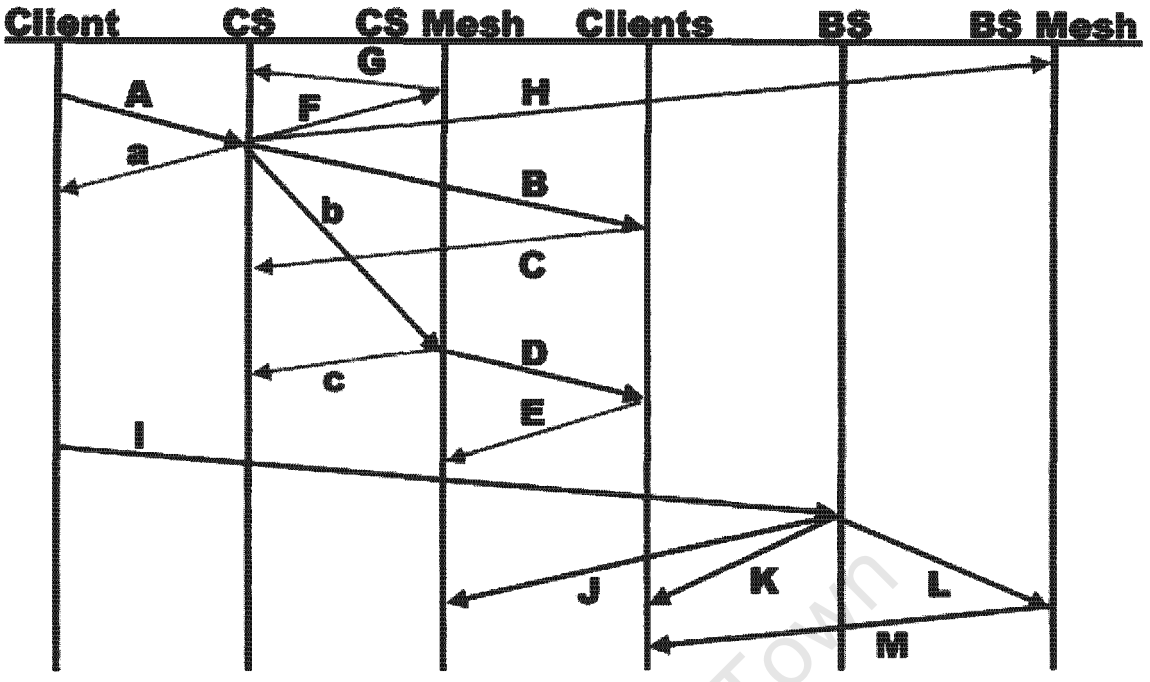


Figure 4.13: A Broadcast and Control Server Receives Data

Key: The *Black* arcs are *Action* arcs

The *Blue* arcs are *Reaction arcs*

The *Green* arcs are *Positive Responses*

The arrowheads direct the flow of data along the arcs between environment entities.

Arcs A and a: The Client sends a Control update to the CS it is connected to. Note that once the client has a connection to the control server, it maintains this connection for the duration of online time. Since control updates are critical to the consistency of the environment, they must be acknowledged and the client will resend the update if it is not acknowledged, as shown by arc a. Note also that there are no negative acknowledgements as the connections have already been established. If the control update requires some sort of negotiation and response, the response occurs within the positive acknowledgment even if the response itself is a denial!

Arcs B, b and D: The CS then distributes the update to all interested clients that are connected to it and passes the update on to the CS mesh. The CSs in the mesh pass the update on to their interested clients and update their own databases.

Arcs C, c, and E: Again, these messages require acknowledgements and the data will be resent if necessary.

Arcs F, G and H: In addition to passing the update on, the CS may choose to recalculate certain Interest Matrices. This will result in an updating of the locally stored interest matrices and the distribution of an update message to the BS mesh, arc G, and the CS mesh, arc F. where the servers in these meshes will update their own stored matrices. Note that the CSs will automatically respond with acknowledgements, arcs H, while the BS will not.

Arc I: This arc shows the client sending a Broadcast Update to the BS it is connected to. No response is required since it is broadcast data and loss of the data can be smoothed by the client if necessary as the data is non-critical.

Arcs K, L and M: The BS will then pass the update on to any connected clients that are interested in the update. It will also pass the data on to the BS Mesh where the BS receiving the data will similarly pass the information on to clients who are interested in the information.

Arc J: Here the BS passes the update on to the CS mesh. Note that this will not occur regularly as it would flood the CSs and is intended to be a 'low granularity' update of the CS as to the position of objects within the environment that may not be producing control updates.

The data that had arrived at the Control and Broadcast servers has now successfully been delivered to the various Client and Environment Servers and the environment has remained consistent.

4.5.4 The Broker Server Request Update Information

The fourth scenario, shown in Figure 4.14, shows a Broker requesting update information from three environment entities. The information, once received, is then distributed to any other Brokers in the Broker Server Mesh. The messages are labelled and described beneath.

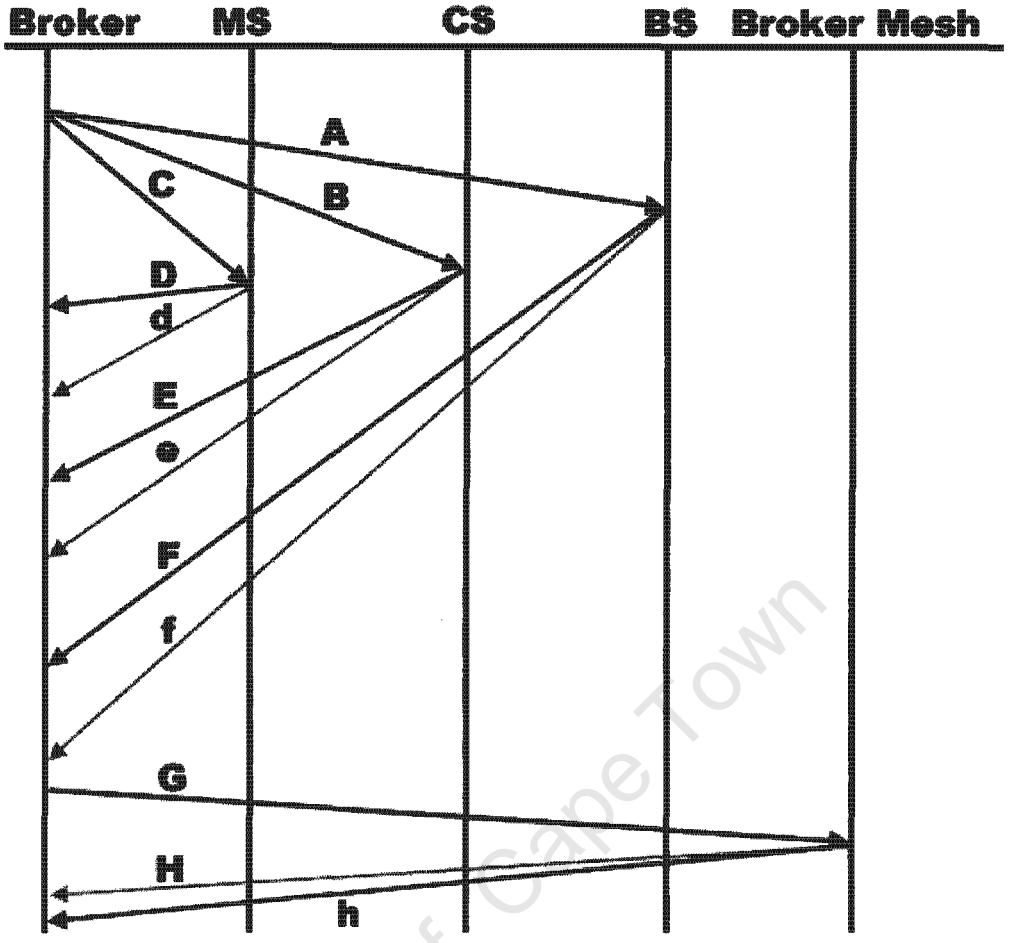


Figure 4.14: Broker Server Requests Update Information

Key: The *Black* arcs are *Action* arcs

The *Red* arcs are *Negative Responses*

The *Green* arcs are *Positive Responses*

The arrowheads direct the flow of data along the arcs between environment entities.

Arcs A, B and C: A Broker, requiring up-to-date information as to the load and activity of the various environment entities, initiates requests to three of these entities, arcs A, B and C. It does this by sending the request to the address, as stored in the Broker database, along with a Server-specific security token. If the entity does not respond, then the Broker can attempt to resend the request. If this occurs again, the Broker will remove the entity from the active list and may attempt to check again later. If this occurs, the Broker will also notify the Broker Mesh of the inactivity of the entity and thus follow arc G.

Arcs D, E and F: Given that the environment server being questioned responds, it will

initially validate the server security token. If it is invalid, it will return a negative response.

Arcs d, e and f: Assuming the security token is valid, the environment sever will respond with its latest Load Index value and any other server specific information. In the case of the MS, the reply may include any new files that the server may have. In the case of the CS and BS, it may include information about the connected user. This information will ensure that the Broker's model and media database is up to date and it will also allow the brokers to maintain records as to client connectivity that can be used for calculation of statistics and billing.

Arcs G, H and h: Assuming the Broker has some information to distribute (an update or a non-response), the Broker will then distribute this information to all other available Brokers in the Broker Mesh. This will ensure that the other Brokers do not request update information from those particular environment entities and flood them, and it will ensure that all the Brokers maintain up to date databases of environment server availability, mode and media availability and their Load Indices. This will allow the Brokers to distribute and balance client requests effectively across the available sever.

When considering the Broker updating its peers, it will send an update to every Broker, shown in arc G, and will resend this update if no response is received. A positive response is received if the sending Broker's security token is valid and the data has been received, arc H, and a negative response is received if the security token is rejected, arc h.

These actions completed, the Brokers will have accurate Load Index values for the various environment entities. This will allow the Brokers to distribute client requests intelligently to the available servers.

4.6 Summary

There are many ways to approach the design of a VE architecture and as noted in the introduction to this chapter, the design process is simply a combination of trade-offs that must be made and supported. The aim of the HEAVEn architecture is to provide a scalable DVE that maintains the response times necessary to ensure a RT interactive and responsive experience for the User and at the same time ensure the consistency of the environment database.

The client-server approach to the DVE architecture enables the design to ensure database consistency and secures specific control related actions that may occur within the environment. At the same time, the individual connections to the clients allow each user to dictate their requirements as long as they fall within the maximum transfer times set out to ensure realistic interaction within the environment.

The various other functional concepts, such as the MS, NS and BS concepts, were added to the design to ensure that it met the functional requirements set out in Chapter 3 while avoiding the limitations and problems encountered by other VE architectures described in Chapter 2.

Chapter 5 Implementation and Test Results

This chapter presents the implementation of the HEAVEn architecture as it is described in Chapter 4. The implemented software is then tested and the results tabulated in the later portion of this chapter.

5.1 Initial Implementation Considerations

The HEAVEn architecture, as described in Chapter 4 and illustrated in Figure 4.10, consists of several distinct components that need to be implemented if a fully functional implementation is to be achieved.

After studying the various logical components in the HEAVEn design and the available resources, it was noted that the only component that needed specific development requirements was the Client. The client implementation uses *Sense8's WorldToolKit* [Sen98] as its rendering engine and its libraries were only available, to us, to be used on a *Windows NT* or *Windows 2000* platform. Further, the libraries were designed to be used in conjunction with *C* or *C++*.

While the Client software is not a focus of this dissertation, a choice had to be made as to what OS platform and programming language to use. To limit any compatibility problems during development, it was decided that it would be best to develop the server and client software on the same platform. Using the various RT extensions to Windows was considered, but was decided against as it was unclear how compatible the rendering engine was with these extensions.

C++ was chosen as the programming language over C due to the object oriented nature of the programming language. This enables simpler code reuse, and should simplify any future attempts to port the software to a RT platform.

5.2 Implementation

The next phase considers each of the logical DVE components. The aim of this section is not to describe in detail how the modules were implemented, but instead to give an overview of how certain of the functions of the modules were implemented, where problems were encountered and how they were overcome.

5.2.1 The Client and 'Tester' Applications

The first application to be considered is that of the Client. Since this application is not a focus of this dissertation, a suitable substitute needed to be developed to allow the testing of the various environment servers. This application is simply called the *Tester*.

The Tester was not designed to be a replacement for the Client software and as such, does not emulate the Client exactly. Instead, it has been designed to produce typical or atypical requests to test the various servers that were implemented. This allows the Tester to be used to test the various *individual functions* associated with the servers, the environment's response to an *individual client* and the environment's response to *multiple clients*. How the Tester tests the various components will be discussed later in this chapter.

5.2.2 The Model and Media, Broker and Name Server Applications

The MSs, BSs and NSs are all servers that offer extra functionality to the connecting client. This functionality enables the client to complete such functions as the location of the environment brokers, the retrieval of the core environment server addresses, secure access to the environment and the retrieval of needed environment model and media files.

The functions offered by these servers are not often used and when they are, a connection is made to the server, a specific task is performed and the connection is ended. While the completion of these tasks is important, the timed requirements of these tasks is unimportant as they either occur before the user's client connects to the environment or in parallel to the client's interactions. In the case of the NSs and BSs, delays will not affect the user's interactive presence as the functions occur before the clients connect to the environment. The interaction with the MSs occurs in parallel to the user's interactions using such methods as pre-fetching and caching and does similarly not affect the user's interactive presence within the environment.

Since the functionality associated with these servers is not time critical and relies only on the functionality completing successfully, the testing of these servers can take place separately from the testing of the BSs and CSs. The test suites used to test these servers can also be exhaustive as there are a limited number of possible correct requests that can occur.

When considering possible results from the testing of these servers, it is important to remember that the servers do not affect the user's RT interaction with the environment and that the client software relies only on the availability of the functionality. Testing is therefore only necessary to ensure that these functions work correctly.

5.2.3 The Control Server

The CSs supply the guaranteed service to the clients necessary to distribute the consistency-critical environment data. This data must be processed and redistributed within the 8ms deadline assigned to the servers in the design. This deadline must be maintained regardless of the number of connected clients or peer-servers. For each control update or request received, the CS must complete one of the following three tasks.

The first task occurs when the update is received from a peer-server. The update is correlated with any contentious updates currently queued at the server that are awaiting ratification. This requires the CS to interrogate each incoming packet of data and this slows the server down compared to the system that will be described and used in the BS. It will then decide which update to apply to the environment database. The update is then applied to the environment database and an acknowledgement sent. The contention resolution process is described more fully in Chapter 4. Once the update has been applied to the environment database, the CS distributes the same update to all interested clients using the interest matrix associated with the source of the data to locate any interested clients. Note that there may be no interested clients connected to the CS and thus no updates will be sent. If an update is sent, the same update is queued awaiting an acknowledgement packet and will be resent later if no acknowledgement is received.

The second task occurs when an update is received from a client. The CS must first check to see if the update will result in a database contention. If it could, it must distribute the data to its peer-servers, queue the contentious update and wait for peer-server responses. If it will not result in a contention, the CS distributes the update to all those connected to it who are interested in the data. As before, there may be no interested clients connected to the CS and thus no further updates need to be sent. The updates that are sent are also queued to await appropriate acknowledgements.

The third and last task occurs when a request for data is received from a client or newly joined CS. The response is the appropriate database portion, encapsulated as a descriptor file, and the positions of the users within the environment segment.

In addition to the processing of received control updates, the CS must also check for any acknowledgement time-outs that therefore require the resending of the data that has not yet been acknowledged. It must also calculate new Interest Matrices for its connected clients. This does not have to occur regularly, but any spare processing time should be used to calculate the interest matrices. The interest matrices, once calculated, will need to be delivered to the peer-servers and to the various BSs.

These processes all need to be completed within the 8ms allotted to the Servers in the environment. The CS must therefore apportion its time so that it is able to handle updates from all connected clients and peer-servers and still have time to resend updates and calculate and distribute interest matrices.

5.2.3.1 Blocking Sockets and Receiving Data

The next concept to consider when implementing the CS is whether to use a blocking or non-blocking socket. A conventional socket represents a connection to the network and has a unique system-wide identifier. This identifier represents the resources allocated by the OS to the socket. In the case of ATM, these resources include the connection's VPI, VCI and any QoS contract that has been negotiated for the connection when it was created. When using sockets, it is important to note that for communications purposes, a sending and receiving operation can only be performed on one socket at a time (i.e. no group socket reading). In addition, the most processor efficient way of using a socket is in what is known as *blocking mode*. In this mode, if an attempt to read data is made and there is data available, the read (or receive) will return immediately with the data. If there is no data available, the function will not return and the thread performing the read will be suspended until data arrives at the socket, at which point it will return. While the thread is suspended, the OS can perform other tasks.

Using a blocking system works fine when considering small-scale servers where only a relatively small number of independent connections will be made to the server, each of which will have a thread associated with it which will continually enter a blocking state waiting for data to arrive on the connection. If data is received by the server irregularly, there will only be a few active receiving threads at any one time.

This system does not scale well, however, as every thread requires system resources and as the number of connecting clients increases, so does the requirement on the system's resources. Further, the above scenario assumes that the data will be arriving irregularly. If data arrives rapidly, many of the threads will be active at once and will thus have to share CPU resources. This will result in the starvation of some of the threads and increase the overall response time of the server to the client data.

Another problem associated with using blocking sockets for the CS and BS implementations is that, when using separate threads and blocking sockets, there must still be a means for communication to occur between the threads. As a solution, messaging could be used to pass data between threads, though this method is particularly slow. Another solution could be to buffer all the outgoing information to the sockets. The first problem with this method is that there must be a means to limit the access to these shared memory buffers to prevent multiple writes. Using mutual exclusion locks prevents multiple accesses, although these locks use extra resources and for 'active' systems, this could result in a long wait before access is granted to the buffers. This will obviously result in extra overall response time delays. The second problem is that the blocking threads will have to receive data before they can access and process the buffered data. This could result in a situation where data collects in the buffer for long periods while the blocking socket receives no data.

5.2.3.2 Non-Blocking Sockets and Receiving Data

A means to solve the above problem is to use the other available socket mode known as the non-blocking mode. In this mode, a thread that performs a read is not suspended if there is no data available. Instead, the read returns with the data if there is data available, or without the data if there is none. While this may seem like the ideal solution, one must note that time is spent in the read process even if data is not retrieved.

Using non-blocking sockets will therefore allow a single thread to step through (or traverse) an entire list of sockets to check for available data. If data exists, it can be dealt with, and if no data exists, the thread simply moves on to the next socket. Since the actions all take place within a single thread, there are no synchronisation issues resulting from multiple thread accesses (which clearly improves system resource usage and overall system efficiency), and since there is only one thread, there are no competing application threads (for system processing time).

There are two major disadvantages to this system. The first is that data arriving at the server will have to, at worst case, wait in the network buffers for a time equal to the traversal time of the entire list,

including any time taken to handle data that may have arrived at other sockets that are being serviced before the socket in question. As the number of users and the quantity of data arriving at the sockets increases, so does the traversal time (assuming the traversal deals with all the data at a particular socket before moving on).

The second problem is, as noted earlier, that checking the socket for information takes time. This potentially useful time is wasted though this will only occur if the socket is checked when there is no data available. In a heavily used system, this efficiency decrease will be less obvious as there will normally be data waiting to be processed. Since the read time is non-zero, this will also affect the time that it takes the access thread to traverse the list, further increasing the latency associated with the receiving of the data.

An example of the above delays could be a system with 100 users with a quarter active at any one time. If the time taken to read data from a socket were 60µs with no time cost for sockets without data, one entire traversal of the socket list, where each attempt reads only one unit of data, would take 1.5ms.

$$(100 * ¼ * 60\ \mu s) = 1.5\,ms$$

Equation 5.1: Time Taken for a List Traversal (Zero 'miss' cost)

If you take the same example with a 20µs penalty for reading from an 'empty' socket, the traversal time would take 3.0ms.

$$(100 * ¼ * 60\ \mu s) + (100 * ¾ * 20\ \mu s) = 3.0\,ms$$

Equation 5.2: Time Taken for a List Traversal (20µs 'miss' cost)

This is clearly double the time taken and while this is only an example, the same effect occurs with the real reading of the data! Initial implementations of the design using this system tested read speeds of approximately 5µs, although the drivers were buggy and any created non-blocking sockets caused a continuous memory leak of about 4kbytes every second of running time! This rapidly caused the software to crash. Once updated drivers were released and obtained, the per-read processing time increased to a staggering 28µs (empirically derived approximation). While this solved the problem of the system crashing, it did significantly reduce the overall performance of the implemented server.

For reference, it is noted that a single ATM send takes in the order of 300μs and thus it is important to consider ways to reduce the quantity of data that a server is sending, as any extraneous data will significantly reduce the effectiveness of the overall system.

5.2.3.3 Sending Data

Up until now, the focus has been the retrieval of data from a socket. For blocking sockets, it is possible to include the sending of data in the same thread used to wait for reads on the socket. This will result in any data that needs to be sent, waiting for new data to arrive. For small-scale systems, this could result in lengthy delays. If multiple threads were being used, the sending thread would have to wait for a lock on the sending queue before sending the data, further increasing the delays.

For non-blocking sockets, the read thread could include a send portion at the end of the read traversal of the sockets. The thread would then traverse the list of output buffers (where the read process has placed any data to be sent) and send the data. This removes any need for exclusion locks as there is still only one thread processing the data being read and sent. Another option is to traverse the input socket list multiple times before completing a single output-buffer list traversal. This allows multiple data packets to be grouped and sent *at one time*, reducing any overheads associated with sending the data and switching between the sending and receiving tasks. While this system seems ideal, the continual traversal of the receiving socket list and sending buffer list are heavily processor intensive. This prevents the application from being useful in non-dedicated environment. While this may seem a problem, it is noted that the servers should be dedicated servers, specific to the environment.

Having considered the socket options and the means to design the input and output cycles for the CS, the following process loop was designed as illustrated in Figure 5.1:

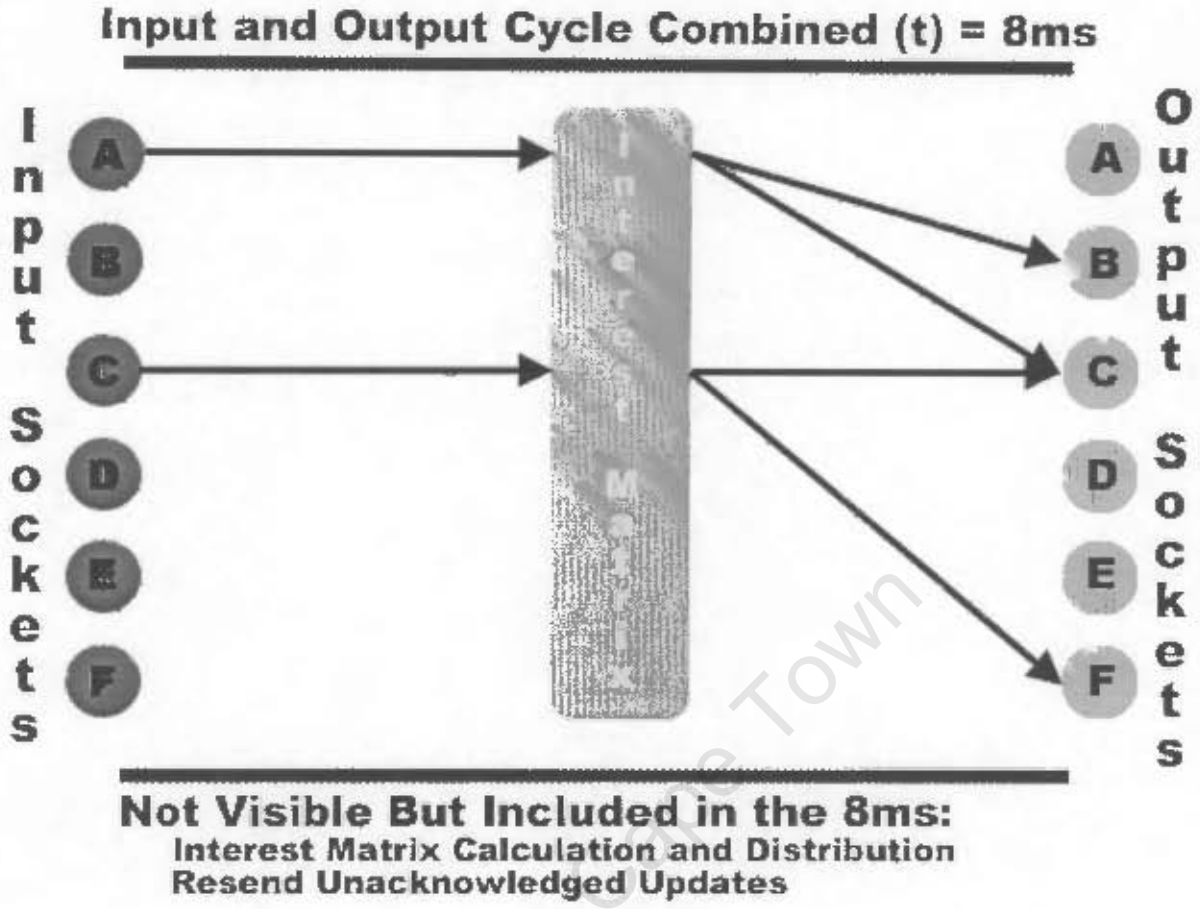


Figure 5.1: The Control Server Process Loop

Figure 5.1 illustrates the currently described process loop used by the CS. Each socket is a non-blocking socket that allows the CS to traverse the input socket list checking for new data. If data arrives, the CS handles the data immediately, sending it, as described above, directly to destination sockets as dictated by the interest matrix. Initially, the use of output buffers was discarded (as can be seen in the figure above) since the control data does not gain from per-connection data collation, which the buffers allow, due to the critical nature of the updates. The removal of the buffers would have also simplified the CS implementation. After implementing and testing the CS as shown in Figure 5.1, it was realised that by removing the grouping of data at the output buffers, we had inadvertently increased the number of send commands that needed to be executed, as a send command had to be completed for every PDU being sent through the server! This severely affected the performance of the server and limited the number of users that the server could support.

Having learnt from the initial implementation's errors, the buffers were added and the implementation can be graphically illustrated as shown in Figure 5.2. Of further interest is that the buffering allows a simple implementation of an acknowledgement system. When data in a buffer is sent, it is not

immediately removed from the buffer. Instead, it is flagged as being 'sent'. When the acknowledgement is received, the data is removed from the buffer. If the time limit for the packet is exceeded, the 'sent' flag is removed and the data will be resent in the next output cycle. To prevent infinite retries, a counter is associated with each element in the buffer. If the sending has not been successful after a limited number of attempts, the update has to be discarded even though this may affect the consistency of the environment.

One drawback of this method is the use of the grouping in conjunction with the AAL5 convergence layer. The grouping and the subsequent packing of the data in an AAL5 frame forces inter-PDU dependence that could reduce the reliability of the environment. Fortunately, the 8ms throughput of the server limits the impact of this problem as the buffers will tent not to have too much data, though this is a generalisation.

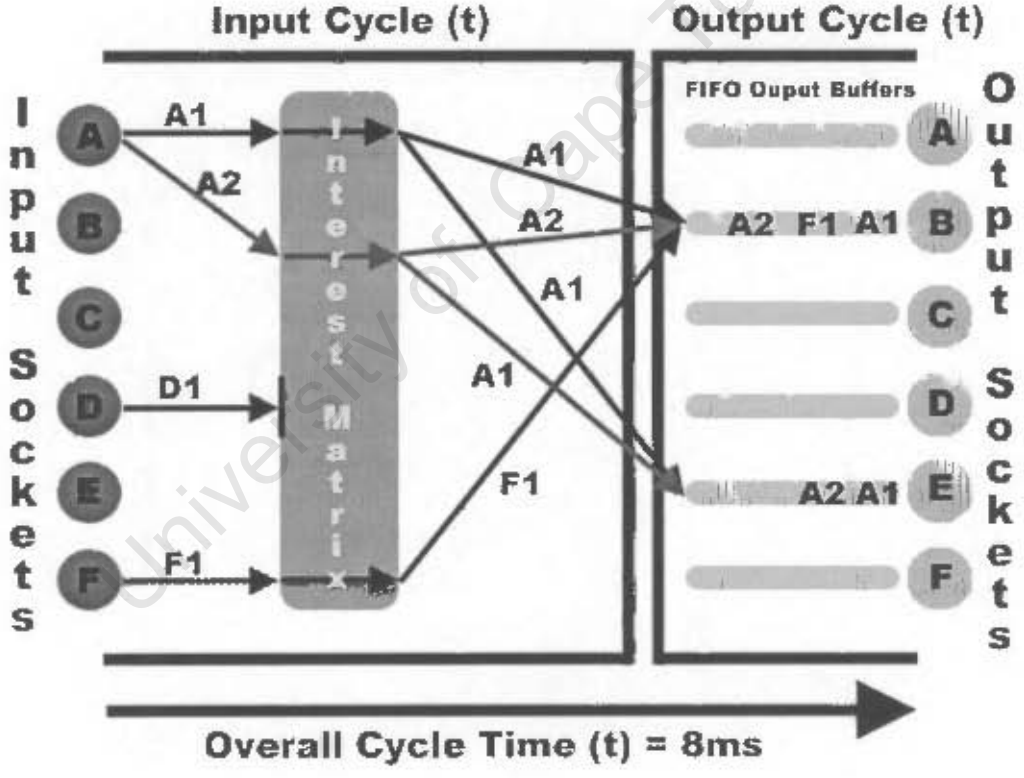


Figure 5.2: The Control Server Process Loop

Figure 5.2 shows the final processing loop of the CS. Of note is the method it uses to divide the 8ms allotted time across the input and output tasks. As an example, follow the data from the various input sockets as the data is read in during the input cycles. Data (A1) is initially received at the first connection (A), applied to the interest matrix and then distributed to the output buffers for connections

B and E. Connections B and C have no data waiting to be read in. Connection D's data, once received, is found to not be of interest by the interest matrix and is discarded. Connection E has no data. Connection F has data (F1) and it is added to connection B's buffer. A second iteration finds only data (A2) at connection A, and the data is placed in buffers for connection B and E. This data (A2) is newer data associated with the same object but must *not* replace the older data as the control information is critical to environment consistency. Note that in this example, we see only the input cycle as it repeats, once the input cycle has repeated its allotted number of times, the output cycle distributes the buffered information. The CS then calculates any interest matrices it needs to and then checks the output buffers for unacknowledged data that must be resent. This data is then flagged for sending in the next output cycle.

While the development of the CS software, other than the slip up with the buffering, went smoothly, one major problem forced a diversion from the designed system. While it was originally thought that Winsock 2 supported VBR services through the ATM adapters, it was later discovered that the VBR service that is offered is non-RT [Mar02]. Since using VBR connections was aimed at guaranteeing the RT requirements of the connection, it was decided that the UBR service should be used to simplify connection creation as no usability statistics were available from which initial bandwidth constraints could be calculated for the nrt-VBR connections or similar CBR connections. The UBR connections were then timed to ensure that they obeyed the RT constraints and the results of this timing are given in the results section of this chapter.

The testing of the CSs focus on how many clients a CS can support. The server is tested by timing the processing loop as it handles data from the various clients. If the server is able to step through all the clients and process their updates and requests within the given 8ms, then the server is able to support the given number of clients at the data rate they are using. The server will naturally be able to support more clients if they are sending less data and this test is therefore a worst-case test. Since no usability data is available, no 'average' cases can be tested. When considering this means of testing, it is important to ensure that the client simulations are producing the control updates and requests at the given rate, as it is possible for the client simulation to become the bottleneck in the testing scenario.

5.2.4 The Broadcast Server

The BSs distribute update data to interested clients within the DVE based on the client's interest matrices. This distribution of the data is a time critical operation that must be completed within the

8ms limit stipulated by the design in Chapter 4. The deadline must be maintained regardless of the number of connected clients or peer-servers or the introduced delays could endanger the interactive experience of the users.

When a BS receives an update, it must distribute the update, regardless of its contents, to every client or peer-server stipulated in the Interest Matrix associated with the data source. Note that if a client is located via another BS, the data is simply sent to the associated server where it will be appropriately distributed.

The implementation of the BS, while requiring that the data be delivered within 8ms, can accept small losses in data and therefore no guarantees have to be made as to the delivery of the data, further simplifying the implementation of the server.

After some initial attempts at implementing the BS, it was decided to implement it in much the same way that the CS is implemented in the discussion above. Since the BS is aimed at receiving and distributing large quantities of temporal data, it was decided to implement the system using non-blocking sockets allowing the main broadcast thread to step through the socket list multiple times. Any incoming data is distributed to an output buffer associated with the appropriate output socket (the appropriate socket is found by using the various interest matrices). The buffer ensures that newer data can replace older data in the output queue thus collating the data sent. Once every 'period', the broadcast thread can send all the data in the output buffers.

Ensuring that the data is sent only once every period reduces the number of 'send' commands that the server has to execute. If the data is sent on a per PDU basis, the number of send commands would be equal to the number of PDUs. Since a single send command takes a significant time, it is in the best interest of the server to limit the number of send commands. The server can do this by grouping data to be sent before sending it. While the grouping of data may seem an ideal solution considering the buffering that is already taking place at the server, it must not be forgotten that the PDUs being sent must remain independent of each other or they will not be able to withstand small losses. This is possible using the AAL0 convergence layer and a PDU size of exactly 48 bytes as the send data will be automatically segmented, when sent, into its constituent PDUs.

The period spoken of above is the 8ms assigned to the broadcast server by the design, and the number of iterations through the input socket list is covered by the time taken to distribute the data in the

output buffers. The time available for reading data in the input cycle is thus the overall cycle time less the time taken to send the data in the output cycle. Any remaining time (less than the 8ms) after the overall cycle is completed, is added to the time available for the iterations of the input cycle. Any time used by the overall cycle that exceed the 8ms limit is removed from the time that is available for iterations of the input cycle. The system thus stabilises itself dependant on the rate at which the data is arriving.

The process loop, as described above, is shown in Figure 5.3 below.

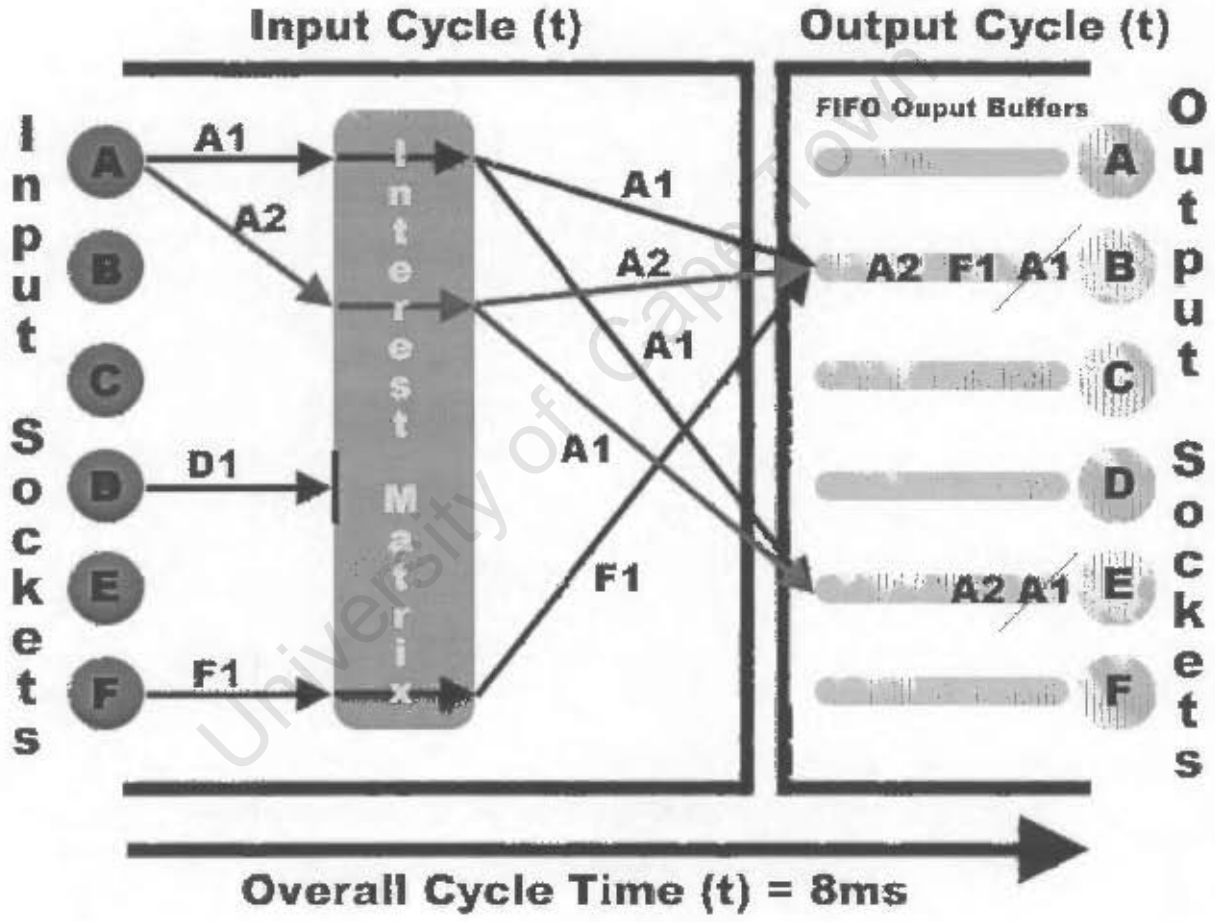


Figure 5.3: The Broadcast Server Process Loop

Figure 5.3 shows the processing loop of the BS. Of note is the method it uses to divide the 8ms allotted time across the input and output tasks. As an example, follow the data from the various input sockets as the data is read in during the input cycles. Data (A1) is initially received at the first connection (A), applied to the interest matrix and then distributed to the output buffers for connections B and E. Connections B and C have no data waiting to be read in. Connection D's data, once

received, is found to not be of interest by the interest matrix and is discarded. Connection E has no data. Connection F has data (F1) and it is added to connection B's buffer. A second iteration finds only data (A2) at connection A, and the data is placed in buffers for connection B and E. This data (A2) is newer data associated with the same object and therefore replaces the older data (A1) in the queue. Note that in this example, we see only the input cycle as it repeats. The output cycle is executed once every 8ms and this will distribute the data in the output buffers to the various destinations.

While the development of the BS software went smoothly, the limitations associated with the creation of VBR connections reduced their usefulness here as they did in the CS implementation. The choice to use UBR connections instead of the nrt-VBR connections was therefore also made here. Without the RT guarantees that rt-VBR offers, the use of the nrt-VBR service class for testing only creates further problems as base traffic specifications need to be calculated where no data is available.

In addition, while the creation of connections using the AAL0 conversion layer seemed to be possible using Winsock 2 and the hardware available, the actual sending of data over these connections was impossible to implement. It was therefore decided to use the AAL5 convergence layer. This compromised the implementation's ability to send cell-sized PDUs as the AAL5 convergence layer packs sent data into an AAL5 frame before sending it. Data is therefore dependent on the other cells being sent. For testing purposes, this problem can be ignored as the network being used is a dedicated network and no errors were recorded during testing. In addition, the overall cycle time of the server is such that the buffers will tend not to fill rapidly though a situation could arise where a single buffer is heavily 'targeted,' becoming a hot-spot, and could result in significant losses should the large AAL5 frame be lost.

The testing of the BS focuses on how many clients a BS can support. The server is tested by first checking that the BS is able to service its input queues at least once within the allotted time. The allotted time is the overall 8ms less the time for a single loop through all the output queues. If this is possible then the server is able to support the number of clients at the given data rate. The server will naturally be able to support more clients if they are sending less data and this test is therefore a worst-case test. Since no usability data is available, no 'average' cases can be tested. When considering this means of testing, it is important to ensure that the client simulations are producing the control updates and requests at the given rate, as it is possible for the client simulation to become the bottleneck in the testing scenario.

5.3 Testing and Results

Once the various applications were created, the testing process took place. While the focus of the dissertation is the design of an architecture to support the requirements for a consistent, real-time, interactive VE, it is important to test and implement design theories and base further development on the results obtained. This section focuses on the results obtained from the testing of the control and broadcast servers. The exhaustive nature of the MS, BS and NS tests do not yield statistical result and is therefore not a focus.

5.3.1 Infrastructure

The test-bed consisted of two personal computers (PCs) (a 450MHx Pentium III and a 1.7GHz Pentium IV) each with 256MBs of RAM and ForeRunnerLe 155 ATM networking interface cards each linked to a central LE155 ATM Switch. Each PC had Windows 2000 Service Pack 3 installed, with Winsock 2.2 and Marconi ForeThought 5.1 drivers installed.

5.3.2 The Model and Media, Broker and Name Servers

As noted in the implementation section, since the functionality supplied by the MS, Broker and NS applications is not time-critical, as long as the applications function correctly and return responses to the client software in a 'reasonable' time, there is no further need to test them.

For the test infrastructure, it is important to note that the initial address resolution takes place through requests sent to the NS. This is followed by environment brokerage and redirection through an environment specific NS. Once this is done, any model and media data can be retrieved, at will, from the MS. For testing purposes, these applications can either be placed on a single machine or spread over multiple machines, as the overheads associated with the once-off use of these applications will not significantly affect the load of even the slowest machine in the test infrastructure. Further, the use of blocking sockets instead of non-blocking sockets ensures that the applications do not dominate the processor time by continually polling the connected sockets. Note also that the functions supplied by these servers tend to occur before the client is fully connected to the environment and therefore do not have RT requirements. If these functions occur while the client is connected to the environment, as in the case of model and media file retrieval, the tasks are not required to be RT in nature and the client software prepares itself to retrieve the data slowly or pre-fetches it before it is needed. The results therefore focus on the Control and Broadcast servers.

5.3.3 The Control and Broker Server

As noted earlier, the testing of the CSs focuses on the CS's ability to support as many clients as possible at a worst-case data rate. The reason a worst-case data rate is being used is that there is currently no information regarding how the client interactions will affect the servers. Usage statistic for an example environment (each environment will have differing server requirement) that uses actual client software and real users to generate the data, is still being prepared at the University of Cape Town [Doy03+]. A similar situation arises when considering a means of testing the BS and as such, the BS is similarly tested for the worst-case scenario, where a number of users connect to it, requiring the server to distribute data at a specific rate.

Given that the client has a maximum of 20ms to complete the rendering of the environment, the retrieval of data from input devices and distribution of an update to the servers, it can easily be calculated than an update is produced every 20ms. Converting this, the client produces 50 updates a second and similarly renders 50 frames of information a second. On the one hand, this may seem excessive considering that only 30 frames need to be rendered every second to produce a realistic display of the environment. On the other hand, when considering that a client may consist of more than just a visual interface to the environment, the other 20 frames could be used for the rendering of sound and other user feedback. Other research [Uda98] has shown that users do not generally produce updates at faster than 20 updates a second, although this research did not consider the many input sources and output devices available to the client.

We therefore have three variables to consider in the testing of these servers. The first is the number of clients that are able to connect to the server, the second is the number of iterations occurring in the input cycle on each of the servers and the third is the size of the interest matrix used.

The number of users that the environment can support is clearly of importance and is a value that can range between no users and infinitely many users. The second variable is of importance as the number of iterations must exceed *one* or the server cannot service all of its input sockets within the allotted input cycle time. If this occurs, a situation could arise where received data waits longer than 8ms before it is served and this would break the time limits for the server interactions as set out in the design chapter.

The size of the interest matrix, the third variable in the testing process, is of importance because the quantity of data that is to be distributed is effectively multiplied by the size of the interest matrix. The data being distributed by a server at any one time is roughly equal to the rate of the updates from the clients multiplied by the number of clients, multiplied by the number of destinations. While the data rate is set at the worst-case rate of 50 updates per second, the number of clients is variable. If every client requests every bit of data sent to the servers, the number of data updates sent would become the square of the number of users multiplied by the data rate. This is therefore a problem of $O(n^2)$ algorithmic complexity. The reason for having an interest matrix is to reduce the number of sources from which a client draws information at any one time. As noted in the design, the choice is based on several algorithms though there is usually an upper bound for the number of sources of data. For the purposes of testing, the number of sources for the interest matrix will be varied though it is important to remember that the bandwidth used by a client with a large Interest Matrix will be high. The real aim of the interest matrices is to reduce the number of sources from which a client draws information and thereby limit the bandwidth used to receive data. If the size of the Interest Matrix is significantly smaller than the number of users, the earlier $O(n^2)$ problem will be simplified to a problem of $O(n)$ algorithmic complexity. While the bandwidth used by the environment is of interest to the overall performance of the environment, it will not be focused on further as the calculating of the bandwidth used for a worst-case scenario is simple. The effect of the size of the interest matrix on the input cycle time is of far more interest if the framework of this dissertation.

The testing of the control and broadcast servers will therefore focus on the server's response, in terms of the number of input cycles, relative to the number of users connected to the server, and the size of the Interest Matrix. As noted in the implementation phase of the chapter, the control and broadcast servers are tested using the Tester application. The Tester application was specifically designed to emulate multiple clients. It was important to ensure that the Tester application was always able to produce data at the required rate (50 updates a second for each client) or the results would not reflect the server's responses correctly. It was found that the Tester, due to its creation of a thread for each client, required significant processor overheads and the acquisition of the 1.7GHz PC was, for this reason, essential in the testing processes. With the Tester situated on this PC, the emulated client processes were able to produce the necessary data.

Under 'normal' circumstances, a Client application would connect to both a Control Server and a Broadcast Server. For testing purposes, since the client simulation can be tasked to send a specific type of data to a specific server, it is possible to isolate either one of these servers to enable individual

testing. This ensures that tests can be rigorously applied without having to interact with another other server. This also simplifies the test procedures, limits the number of PCs that need to be used in testing and prevents any problems that may arise from inter-server conflicts.

In addition to the above test procedures, the length of time that data took to travel between environment components was measured. While this measurement is not critical to the environment, it does allow us to ensure that the UBR connections, used in place of the rt-VBR connections, are maintaining the delivery guarantees set out in the latency and delay framework in the design chapter.

5.4 Results Obtained

With the infrastructure in place, the testing of the Broadcast and Control servers began with the Tester application being run on the 1.7GHz machine. Either the BS or the CS was run on the 450MHz machine and results were obtained from the interactions between the simulated clients and the appropriate server. Each simulation, which corresponds to a point on the graph, is a specific combination of the number of connected users and the maximum interest matrix size, which yields a number representing the average number of input cycles completed by the servers. Note that the number can be a real number as this represents partially completed cycles. This value represents the ability of the server to continue functioning correctly with the current client and data load. When the server can no longer complete an entire input cycle (and thus the overall cycle within the stipulated 8ms), it is considered to have failed. This means that it can no longer support the associated data load and thus the number of connected clients. The server would then have to reduce its client load. Each simulation was run for 10 minutes and each graph represents approximately a day's computation.

While the time taken by the overall server processing cycle was measured, the results are predictable and fall within the time limits set out in the designed latency framework (i.e. Since the system self-corrects to approximately 8ms, the average overall cycle time is 8ms). The servers will only break this time limit when it is unable to complete an entire input list traversal, at which point the input cycle count will indicate the failure of the server to complete its task. This can be seen when the number of input cycles is less than one.

The following graphs were obtained from the simulations for the Control and Brokers servers. The results are shown in Figure 5.4 and Figure 5.5 and are both discussed further.

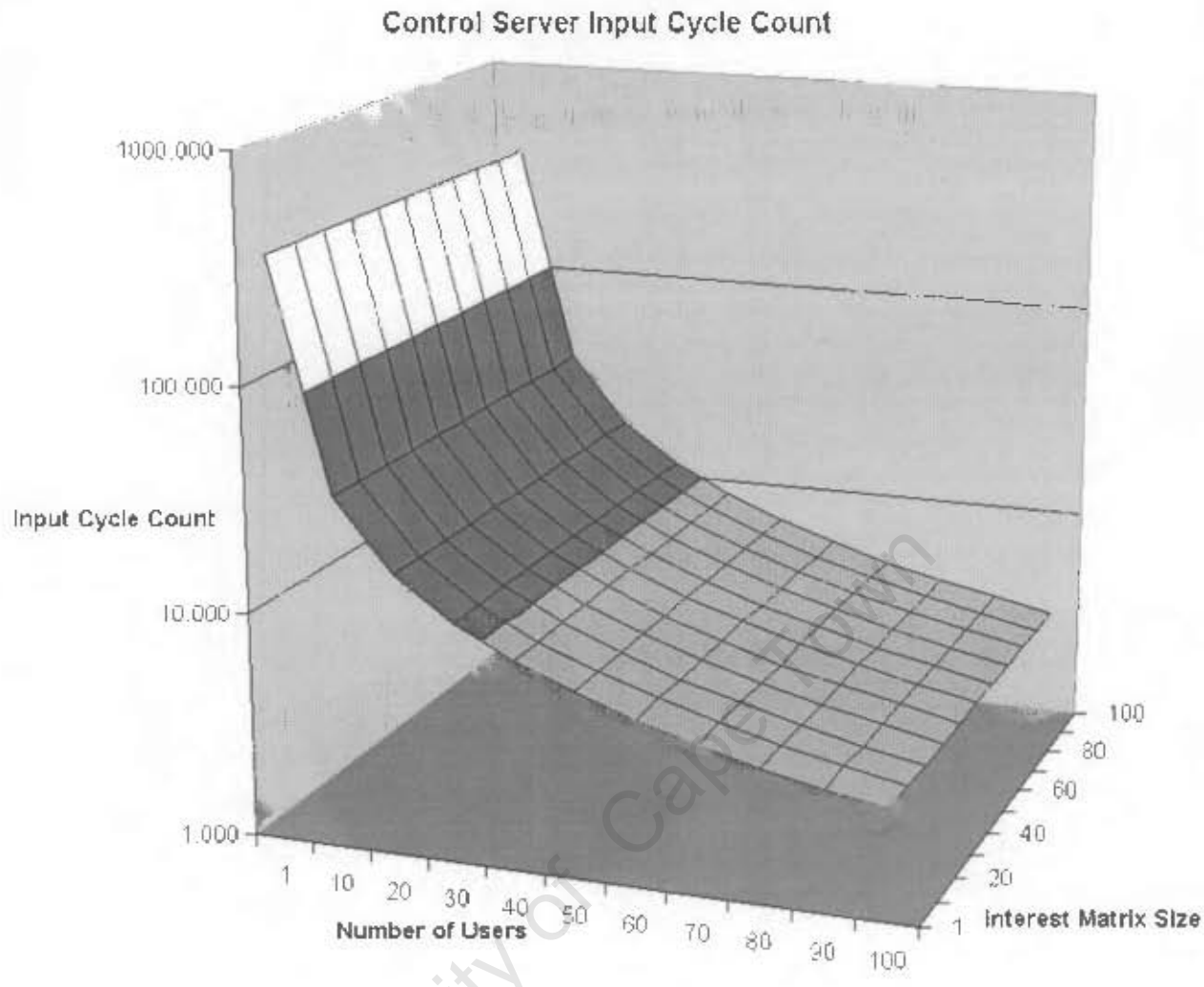


Figure 5.4: Results for the Control Server

Figure 5.4 shows the CS's ability to support connected clients. The first point to note is that, as the number of users increases, so the number of input cycles decreases. This is relatively self-explanatory as the length of the output and input cycles are directly proportional to the number of input sockets that must be checked and read from and to the number of output buffers (one for each output socket) that must be checked and any data sent. When more clients connect, there are simply more sockets and buffers that must be checked each time the appropriate cycle (input or output) occurs and thus fewer cycles can be completed in the allotted time.

The second point of interest is that there is no change in the input cycle time as the interest matrix size increases. This may seem strange when considering that the size of the interest matrix is relative to the quantity of data that must be distributed and thus should intuitively increase the output cycle time and thus reduce the input cycle time. While this may be intuitive, it is important to realise that *all* the data in an output buffer is sent in a *single* send command. As the interest matrix increases, so will the

quantity of data that needs to be buffered for each output socket (as there are more sources of data supply each buffer). As long as the output buffers are large enough to hold the data, the data will all be sent and sent in a single data send. Therefore, the number of send commands per output cycle remains constant.

A third point of interest is to consider how the testing can be extended. If the size of the interest matrix for each client is increased (further depth in the graph – y-axis), the server will eventually fail due either to quantity of data that would need to be stored in the limited output buffers, or due to the maximum available bandwidth of the network connections being reached. If the number of users is increased (further width to the right – x-axis), the server will eventually fail when the input cycle time cannot be completed within the allotted time. For 100 users, the server still completes at least three full input cycles.

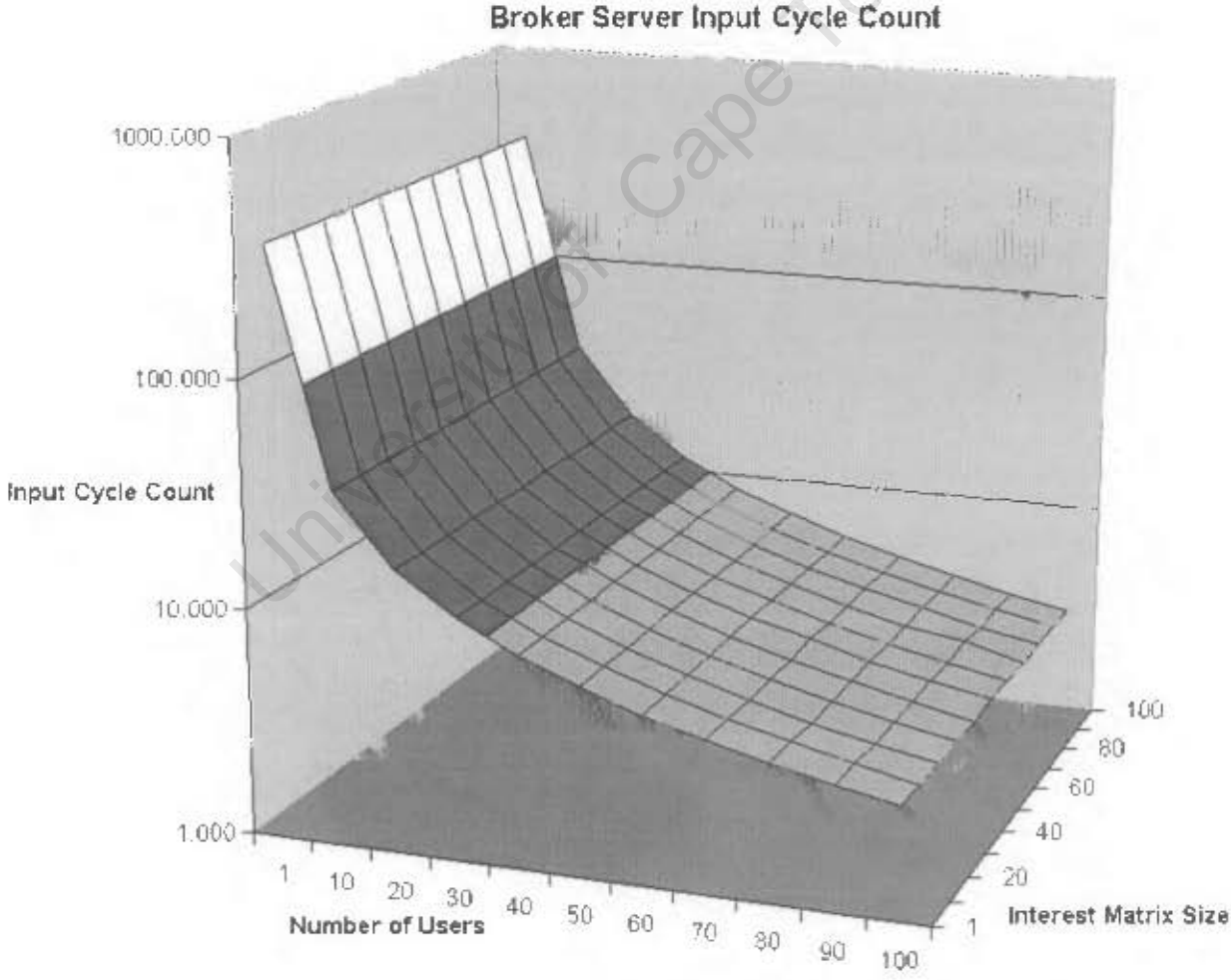


Figure 5.5: Results for the Broadcast Server

Figure 5.5 shows the simulation specific to the Broadcast Server. While it might seem strange that the above graph looks similar to the graph in Figure 5.4, it is noted that the two servers function almost

identically and therefore have the same characteristic responses as described above. The differences between the two servers lie in the implementation of the acknowledgement protocol for the CS and the per connection data collation for the BS. It was initially suspected that these two factors would have an impact on the server's ability to support clients, as some of the server's processing time would be taken up with the acceptance and sending of the acknowledgement PDUs (in the case of the CS) or the source checking and collation of data (for the BS). While these were the original assumption, on testing it is obvious that these factors have little on no effect on the input cycle time.

On closer examination, it was discovered that the data collation was actually of little use in the design as it stands. Collation of data takes place at the output buffer for a specific socket associated with a BS. If data is received from the same source as the data in the queue, it will collate this data, replacing the older with the newer data. While this may seem like a useful mechanism to have in place in the BS, when one realises that the BS's overall cycle time (8ms) is less than half the time taken for a client to create a single update (20ms) , it is understandable that the server should never be able to collate the data being sent!

In the case of the acknowledgement protocol associated with communication in the CS, it was noted that sent acknowledgements were added to the output buffers and while they affected the quantity of data being distributed, they did not adversely affect the input cycle time due to microsecond delays that the acknowledgement PDU creation added. When considering acknowledgements being received, it is noted that the time difference between a read that returns a 'pass' and a read that returns data is also minimal. The receiving of acknowledgements does therefore also not affect the input cycle times.

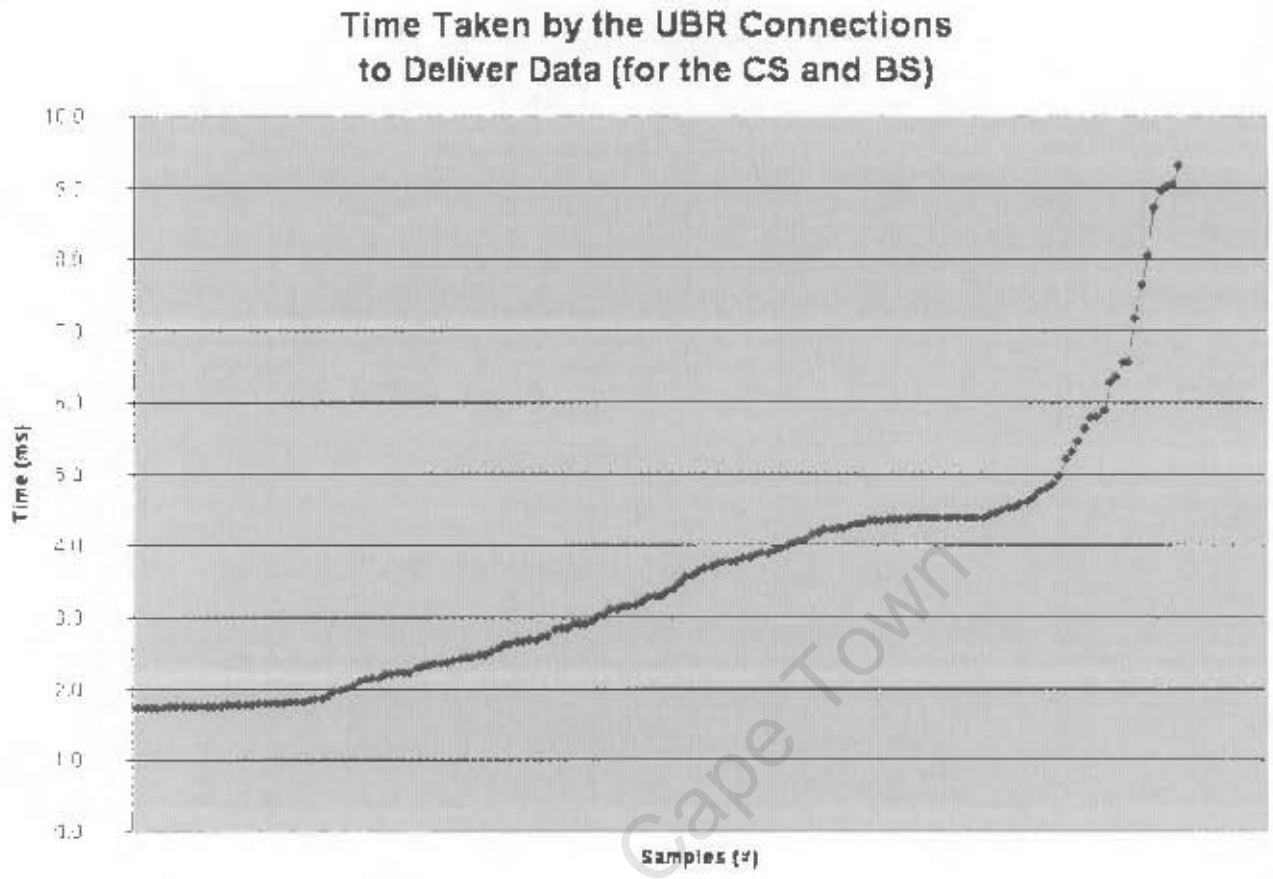


Figure 5.6: Timed UBR Connection Data Delivery

Figure 5.6 is an addition to the above results and is here as a means of confirming that the UBR connections supplied the necessary timed delivery of the environment data in the place of the rt-VBR connections. It is noted that the UBR connections were only able to supply this service because they were used over a dedicated ATM network. Figure 5.6 is an ordered scatter graph showing the connection times for the UBR connections used in both the BS and CS simulations. As can be seen, all of the UBR connections used transported their data within the 24ms upper bound listed in the latency framework of the design chapter. In this, the UBR connections did not affect or hamper the testing process or the results obtained.

Chapter 6 Conclusion and Recommendations

Having designed and implemented the HEAVEn distributed virtual environment platform, a number of conclusions can be drawn that highlight the reasoning for certain design decisions. Recommendations for future work on this topic are also noted.

6.1 Conclusion

While the design and implementation of a RT interactive and scalable DVE using ATM as its networking protocol was itself the aim of this dissertation, several other conclusions can be made.

6.1.1 ATM is a suitable networking technology for supporting a DVE

ATM confers several useful characteristics to its connections and offers several useful services that are all important consideration when creating a reliable RT interactive DVE.

The first characteristic is that ATM cells, if successfully delivered, will arrive in order. This reduces the overheads at both the clients and the servers associated with the reordering of the data, as ordered data is critical to ensuring a consistent view of the environment.

The second characteristic is the cell size. The limited cell size ensures that the per cell data overheads are limited when considering the communication in the environment tends to rely on close to cell size PDUs to distribute the data. The further use of the AAL0 convergence layer will allow cell sized PDUs to be sized at 48 bytes rather then the 43 allowable bytes when the AAL5 convergence layer is used.

The use of the rt-VBR QoS specifications could allow the overall DVE to ensure that communication takes place within the RT interactive response time limits if the design was implemented on a non-dedicated network. This is clearly the basis for the HEAVEn design and the fact that it could not be implemented was a serious drawback.

The use of an ATM network in this dissertation, while prescribed, is a networking protocol that adds much to the usability and reliability of a DVE designed specifically to use the services supplied by the ATM network.

6.1.2 Multicasting is not as beneficial as initially thought

While multicast connections may seem to be particularly useful in the broadcasting of streamed media, in the design of a DVE it is not as useful when considering that the DVE must be scalable. To ensure scalability, the quantity of data that is being produced must be limited before sending it to the clients and servers. This requires individual connections to users, as it is not often that two users will require exactly the same data.

The use of a multicast group per environment object is an interesting option, especially when considering using an IP network, but the lengthy delays in an ATM network when joining a multicast group limit the usability of this idea.

6.1.3 VEs have strict end-to-end delay requirements that must be maintained

VEs have strict delay limits regarding data being sent and received. If delays are longer than 100ms for an immersive VE, then the user's belief in the realism of the environment will be broken and the user could feel some discomfort.

The delays must not only be considered in terms of the network, but also in terms of the applications, servers, operating system and communications stack. Often these components can introduce random delays that significantly affect the realism of the VE. Further, if a client-server model is used, information will have to travel further, often doubling the round trip time.

If the delays, associated with the various components in the path of travelling data, can be guaranteed, it is possible to construct a latency framework that will ensure that the DVE remains RT and interactive

It should be possible to guarantee the delivery of the data required by the framework by using an ATM network's rt-VBR connection thus leaving the remaining guarantees to the applications and the OS on which the applications are running.

6.1.4 Data within an environment must be limited to ensure scalability

To ensure the scalability of the DVE, it is essential to limit the quantity of data that must be handled by the central servers or they will become bottlenecks. In addition, without the limiting of the data, the clients themselves will not be able to handle the quantity of data that they will be receiving.

The HEAVEn design uses two systems to limit the data. The first is the interest matrices that specify the interest of a specific client in the source of the data. The second is the per-connection data collation that occurs in the broadcast servers and removes older data in favour of any newer data that may have arrived regarding an object.

6.1.5 Descriptor files ensure that environment data is not affected by model downloads

While a DVE can require many data files, and these files can become rather large, the responsiveness of the environment is not affected by the downloading of these files due to the ability of the clients to pre-fetch and cache model and media data from the MSs. Since this data is separate from the control and broadcast servers, the retrieval of this data does not adversely affect the functions supplied by these servers, and the use of the test based descriptor files ensures that the data files can be easily referenced and retrieved while limiting the impact, on the CSs, of storing and distributing the environment database.

These descriptor files are also hierarchically organised to allow the client to set the LoD when viewing the environment. By only viewing the higher levels of models in the descriptor file, the client can limit the processing power needed to display the VE while also limiting the model information that is needed.

A further advantage of the descriptor file's hierarchical nature is that clients can slowly build the view of the VE. This does not force the client to download all the model data before viewing portions of the environment. This also aids in continual development of the VE as new areas in the environment can be easily added by adding new entries to the descriptor files and ensuring that the model and media files are available on Model and Media Servers.

6.1.6 The separation of control and broadcast data is justified

Control information for a VE consists of data that must be delivered as it consists of major changes or access information and is critical to the consistency of the environment. Broadcast data is update information about small changes in the environment and should data be lost, it can easily be smoothed over using client-side rendering algorithms.

By separating the control and broadcast data and sending it over separate connections, the DVE can use connections with different QoS requirements to guarantee the delivery of the control information while still offering variable levels of broadcast service.

6.2 Recommendations

The recommendations focus on areas of interest for future research into this topic. As this dissertation was designed as a research platform into DVEs over ATM networks, the work that can still be done is aimed at research into the usability, reliability and characteristics of the DVE created.

6.2.1 Implement the servers as application on a RT OS or within the RT kernel

Due to the stringent end-to-end delay requirements noted in design section of this dissertation, there is the need to guarantee the server performance. The only possible way to do this is to implement the servers in a RT environment. There are two options to consider. The servers can be implemented as software applications that run on the RT platform and specify their needs to the OS, or they can be implemented as part of the OS kernel itself. Both options allow the inclusion of hard RT deadlines for aspects of the processing. Implementing the servers as part of the OS kernel allows for the bypassing of several of the applications interface layers that the OS provides and thereby reduces the time overheads associated with processing the data. The implementation of the servers in the OS kernel is a concept explored by the MASSIVE project and it has shown positive results [Gre99a].

6.2.2 Investigate the Usability Statistics

One of the limitations of the current implantation is that it cannot realistically predict the data that will be sent by the clients to the environment servers. This resulted in the use of worst-case simulations to test the servers. If usability statistics were available, the environment could be tested using this information and thus more accurate predictions as to the ability of the servers to support clients could be found.

6.2.3 Investigate the use of Active Networks to Enhance Multicast Distribute of Updates

An *Active Network* is a network architecture in which the network nodes are able to perform customisable computations on the data being switched or routed [Ten97] [Pun00]. When considering a multicast connection through an active network, it is conceivable that at each node in the multicast tree, a decision can be made as to the necessity of the data further down the multicast tree. If it is not needed, it is discarded. This is illustrated in Figure 6.1.

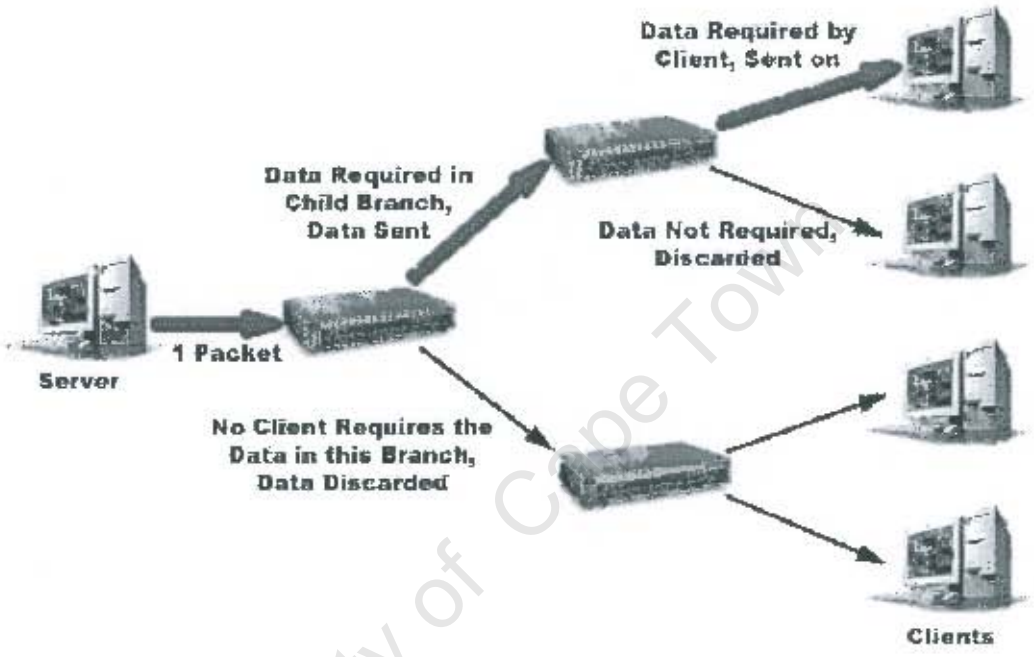


Figure 6.1: Intelligent Network Nodes

Each server hosts an intelligent multicast tree to which the server's clients are connected. Once the server has received an update and checked if it is acceptable, it sends it to the multicast tree leaving the tree to decide if the data is to be discarded or not. Once the data reaches the network nodes, it is either discarded or passed on dependant on the needs of the next node in the multicast tree. The needs are based on a combination of the needs of all the clients connected to the leaf nodes of the multicast tree and thus data is slowly pruned when it is no longer needed. The active network's multicast tree is capable of supporting different QoS requirements by simply pruning the data before sending it to the user, and similarly is able to guarantee delivery latency by ensuring the maximum time to send the data to the client is smaller than the latency limits.

Further, simple AoI calculations could be made at the active network nodes and while this would increase the processing overheads, it reduces the bandwidth and congestion that could occur in the network, as more data pruning will be done. To facilitate this, the clients would need to be able to

communicate their position and interest in data to the network nodes and this would likely be done via the server to ensure security.

6.2.4 Investigate compression techniques for model and media data

Environment data files can become rather large and this can be a problem for both the clients and the servers storing these files. Compression techniques should be investigated with the aim of finding a technique that does not require large processing overheads as the clients cannot be expected to be able to cover these overheads. The optional distribution of compressed data can be considered although this will require that either the MSs store a compressed and uncompressed version of every file, or compress files on-the-fly before sending. The use of these compression techniques would however, reduce bandwidth needed to transfer the data files.

6.2.5 Further testing

The DVE server platform has been designed and implemented and can now be used to test the design further. Testing can still cover several areas including the response of the architecture to increased latencies, increased data loss and the introduction of more servers and clients. The collection of further data to support the usability of the architecture and the bandwidth and processing needs is essential if the system is to be of commercial use. Usability testing to ensure that the architecture is supporting an immersive VE experience is similarly required and much of this testing is still in progress at the Communication's Research Group [Doy03+].

References

Key	Reference
[4xT02]	4x Technologies. *Phoenix3D.* 4x Technologies. 2002. Site: http://www.4xtechnologies.com
[Ahm02]	Ahmad, Khalid. *Sourcebook of ATM and IP Internetworking.* IEE Press, Piscataway, USA. 2002. ISBN: 0-471-20815-9
[Ahn97]	Ahn, Jesung; Lee, Minjeong; Lee, Huenjoo. *DOOViE: An Architecture for Networked Virtual environment Systems.* Information Technology Laboratories, LG Corporate Instiute of Technology, Korea. Proceedings of the Computer Graphics International. 1997.
[And95]	Anderson, D.B; Barrus, J.W; Howard, J.H; Rich, C; Shen, C; Waters, R.C. *Building Multi-User Interactive Multimedia Environments at MERL.* IEEE Multimedia Volume 2, No 4. 1995.
[Ari99]	Arikawa, Masatoshi; Shimojo, Shinji; Amano, Akira; Maeda, Kaori; Aibara, Reiji; Nishimura, Kouji; Hiraki, Kaduo; Fujikawa, Kazutoshi. Real-Time Spatial Data Management for Scalable Networked Augmented Virtual Spaces. IEICE Transactions. Volume E82-D, No 1. January 1999.
[Asc01]	Ascension Technology Corporation. *MotionStar.* Ascension Technology Corporation. 2001. Site: http://www.ascension-tech.com/
[Ash02]	Microsoft Corporation. *Asheron's Call.* Microsoft Corporation. 2002. Site: http://www.microsoft.com/games/ac/ Site: http://www.microsoft.com/games/acdm/
[ATM00a]	The ATM Forum. *Frame-based ATM Transport over Ethernet (FATE).* The ATM Forum - Technical Committee. Worldwide Headquarters, California. February 2000.
[ATM00b]	The ATM Forum. *Frame Based ATM over SONET/SDH Transport (FAST).* The ATM Forum - Technical Committee. Worldwide Headquarters, California. July 2000.
[ATM00c]	The ATM Forum. *ATM Name System v2.0.* The ATM Forum - Technical Committee. Worldwide Headquarters, California. July 2000.
[ATM00d]	ATM Forum. *ATM User Network Interface (UNI) Signalling Specification Version 4.0.* af-sig-0061.000. The ATM Forum - Technical Committee. Worldwide Headquarters, California. 2000.
[ATM01]	The ATM Forum. *Beginners' Overview of Asynchronous Transfer Mode.* The ATM Forum. 18 September 2001. Site: http://www.atmforum.com/pages/aboutatmtech/beginnersguide.html The ATM Forum. *The History of ATM.* The ATM Forum. 18 September 2001. Site: http://www.atmforum.com/pages/aboutatmtech/atmhistory.html
[ATM02]	The ATM Forum. *The ATM Forum.* The ATM Forum. 2001 Site: http://www.atmforum.com/
[Bar01]	Barrass, Hugh. *Ethernet over Copper.* Cisco Systems. 11 March 2001.

Key	Reference
[Ber01a]	Berkley, Jeffrey. *Suture Simulation Based on Fast Finite Element Analysis.* The Human Interface Technology Laboratory at the University of Washington, 2001. Site: http://www.hitl.washington.edi/projects/suturesum/
[Ber01b]	Bernier, Yahn W. *Latency Compensation Methods in Client/Server In-game Protocol Design and Optimisation.* Kirkland. 2001.
[Bil99]	Billingham, Mark; Kato, H. Shared Space. The Human Interface Technology Laboratory at the University of Washington, United States of America and ATR Media Integration and Communication, Kyoto, Japan. 1999. Site: http://www.hitl.washington.edu/projects/shared_space/
[Boe02]	The Boeing Company. *Boeing.* 2002 Site: http://www.boeing.com/ The Boeing Company. *Phantom Works – Shaping the Future of Aerospace.* 2002. Site: http://www.boeing.com/phantom/pw_sfa.html
[Bon01]	Bonaventure, Olivier; Nelissen, Jordi. *Guaranteed Frame Rate: A Better Service for TCP/IP in ATM Networks.* University of Namur and Colt Telecom, Belguim. 2001. Disk: \Papers\Networking\Garanteed Frame Rate - A better service for TCP-IP in ATM networks (2001).pdf
[Bra95]	Bradner, Scott. *Recommendation for IPNG - RFC1752.* Harvard University. Cambridge. January 1995.
[Bud02]	Budd, Timothy A. *An Introduction to Object-Oriented Programming. Third Edition.* Addison-Wesley Co. 15 January 2002. ISBN: 0-201-76031-2
[Cas98]	Casher, Omer; Leach, Christopher; Page, Christopher S; Rzepa, Henry S. *Virtual Reality Modelling Language (VRML) in Chemistry.* Chemistry in Britain. 1998. Site: http://www.ch.ic.ac.uk/rzepa/vrml
[Cat02]	NCSA, Board of Trustees of the University of Illinois. *Caterpillar's Distributed Virtual Reality Project.* NCSA, Board of Trustees of the University of Illinois. 2002. Site: http://archive.ncsa.uiuc.edu/VEG/DVR/ Caterpillar. *Caterpillar.* 2002. Site: http://www.caterpillar.com/ NCSA, Board of Trustees of the University of Illinois. *NCSA Recognises Caterpillar for Partnership That Spawned a Series of Technology Breakthroughs.* NCSA, Board of Trustees of the University of Illinois. 2002 Site: http://www.ncsa.uiuc.edu/News/Access/Releases/020507.GCA.html
[Cat98]	Caterpillar. Caterpillar's Distributed Virtual Reality. Caterpillar. 1998. Site: http://archive.ncsa.uiuc.edu/VEG/DVR/dvr02/sld001/htm
[Cis99]	Cisco System Inc. *Internetworking Technologies Handbook - Chapter 27: Asynchronous Transfer Mode.* Cisco System Inc. June 1999. Site: http://www.cisco.com/univercd/cc/td/doc/cisintwk/ito_doc/index.htm Cisco Systems. *Internetworking Technology Overview.* Cisco Systems. 1999.

Key	Reference
[Cod02]	Code Cult. *Code Creatures*. Code Cult. 2002. Site: http://www.codecult.com
[Coh94a]	Cohen, D. *NG-DIS-PDU: The Next Generation of DIS-PDU (IEEE-1278).* Proceedings of the 10th Workshop on Standards for Distributed Interactive Simulations. March 1994.
[Coh94b]	Cohen, D. *DIS - Back to basics.* Proceedings of the 11th Workshop on Standards for Distributed Interactive Simulations. September 1994.
[Cov96]	COVEN. *COllaborative Virtual ENvironments.* COVEN. 1996. Site: http://chinon.thomson-csf.fr/projects/coven/
[Cub98]	Cubeta, James A; Kern, Kirt T; Egts, David D; Oberysekara, Upul. *VENoM - Virtual Environment for Network Monitoring.* Center for Computation Science, Naval Research Laboratory, Washington, DC and Concurrent Technologies Corporation, Johnstown, PA. 1998.
[Dee89]	Deering, Steve. *Host Extensions for IP Multicasting - RFC 1112.* Computer Science Department. Stanford University. August 1989.
[Def02]	Defence Modelling and Simulation Office. *Defence Modelling and Simulation – HLA.* Defence Modelling and Simulation Office. 2002. Site: https://www.dmso.mil/public/
[Del02]	Delaware Biotechnology Institute. Accelerating Life Sciences Research through Visualisation. Delaware Biotechnology Institute, University of Delaware and Silicon Graphics Inc. August 2002. Site: http://www.sgi.com/features/2002/aug/dbi/
[Dia97]	Dias, J.M.S; Galli, R; Almeida, A.C; Bello, C.A.C; Rebordao, J.M. *mWorld: A multiuser 3D virtual environment.* IEE Computer Graphics and Applications, Volume 17, No 2. Mar-April 1997.
[DIV02]	Swedish Institute of Computer Science. *The Distributed Interactive Virtual Environment (DIVE).* Swedish Institute of Computer Science. 2002. Site: http://www.sics.se/dive/
[Doy03+]	Doyle, Stuart. *The Design and Implementation of a DVE over ATM Networks.* The University of Cape Town, South Africa. *An unfinished Master's Dissertation.*
[DSL01]	The DSL Forum. *ADSL Tutorial.* The DSL Forum. 2001 Site: http://www.dslforum.com/
[Dun02]	Dunningan, James. *Gameboys at War.* The Strategy Page. 13 May 2002. Site: http://www.strategypage.com
[Ecl02]	Eclipse Entertainment. *Genesis3D.* Eclipse Entertainment. 2002. Site: http://www.genesis3d.com/
[Eve02]	Sony. *EverQuest.* Sony Computer Entertainment America Inc. Sony Online Entertainment Inc. Sony Computer Entertainment America Inc. 2002. Site: http://eqlive.station.sony.com/ Site: http://everquest.station.sony.com/
[EVL02]	Electronic Visualization Laboratory. *The CAVE: A Virtual Reality Theater.* University of Illinois at Chicago. 2002. Site: http://www.evl.uic.edu/pape/CAVE/

Key	Reference
[Fab00]	Fabre, Yoann; Pitel, Guillaume; Soubrevilla, Laurent; Marchand, Emmanuel; Geraud, Thierry; Demaille, Akim. *A Framework to Dynamically Manage Distributed Virtual Environments.* EPITA Research and Development Laboratory, France. Proceedings of the 2nd International Conference on Virtual Worlds. July 2000.
[Fak02]	Fakespace. *The Cave.* Fakespace. Ontario, Canada. 2002. Site: http://www.fakespace.com
[Fla01]	Flaherty, Nick. *Video to get surround treatment with photorealistic images.* Electronic Times. 19 September 2001. Site: http://www.electronictimes.com/story/OEG20010919S004 Site: http://www.electronictimes.com/printableArticle?doc_id= OEG20010919S004
[For98]	Fore Systems. *ForeRunner LE Adapters.* Fore Sytems. 1998.
[Fuj98]	Fukikawa, Kazutoshi; Tomohiro, Taira; Oh, Seiwoong; Kado, Daisuke; Shimojo, Shinji; Miyahara, Hideo. *A Quality Control Mechanism for Networked Virtual Reality Systems with Video Capability.* Osaka City University, Osaka, Japan (and others). 1998.
[Fun95]	Funkhouser, T. *RING: A Client-Server System for Multi-User Virtual Environments.* ACM SIGGRAPH Interactive Graphics, Monterey, California. April 1995.
[Gib00]	Gibson, Simon, Howard, Toby; Murta, Alan. *Virtual Environments for Scene of Crime Reconstruction and Analysis.* In Proceedings of SPIE Electronic Imaging 2000, volume 3960, San Jose. January 2000 ISBN: 0-8194-3578-3 Gibson, Simon; Hubbold, Roger; Howard, Toby; Murta, Alan; Oran, Dan; Sinnott, James. Reconstructions from Video of Environments with Accurate Lighting. Department of Computer Science, Manchester University, England. 2002. Site: http://aig.cs.man.ac.uk/research/reveal/icarus/index.html
[Gib85]	Gibson, William. *Neuromancer.* Ace Books, New York. 1985. ISBN: 0-441-00068-1
[Gre95]	Greenhalgh, Chris; Benford, S. *Massive: A Distribute Virtual Reality System Incorporating Spatial Trading.* Proceeding of the 15th International Conference on Distributed Computing Systems, Vancouver, Canada, IEEE Computer Society Press. 1995.
[Gre99a]	Greenhalgh, Chris. *Massive-3 / HIVEK Introduction.* Department of Computer Science, University of Nottingham, Nottingham, UK. June 1999. Disk: \Papers\Recommendation\ Massive-3 - HIVEK Introduction.pdf
[Gre99b]	Greenhalgh, Chris. *Analysing Awareness Management in Distributed Virtual Environments.* Department of Computer Science, University of Nottingham, Nottingham, UK. June 1999.
[Har00]	Harm, Deborah L. *Motion Sickness Neurophysiology, Physiological Correlates and Treatment.* 2000. Site: http://vehand.engr.ucf.edu/handbook/Chapters/Chapter36.html
[HGP02]	The Human Genome Management Information System. *The Human Genome Project.* Oak Ridge National Laboratory. June 2002. Site: http://www.ornl.gov/ Project: http://www.ornl.gov/hgmis/

Key	Reference
[Hof02]	Hoffman, H.G; Doctor J.N; Patterson, D.R; Carrougher, G.J; Furness, T.A. III. *VR Pain Control.* Human Interface Technology Lab. 2002. Site: http://ftp.hitl.washington.edu/research/burn/
[Hol02]	Holbrook, Hugh; Singhal, Sandeep. *The PARADISE Project.* The Distributed Systems Groupd, Stanford University. Site: http://www.dsg.stanford.edu/paradise.html
[Hon96]	Honda, Yasuaki; Matsuda, Kouichi; Rekimoto, Jun; Lea, Rodger. *Virtual Society: Extending the WWW to support a Multi-user Interactive Shared 3D Environment.* Sony Computer Science Laboratory, Tokyo, Japan. 1996.
[IEE00]	IEEE Society. *Carrier sense multiple access with collision detection (CSMA/CD) access method and physical layer specifications - IEEE Standard 802.3: 2000 Edition.* The Institute of Electrical and Electronics Engineers, Inc. New York, USA. 16 October 2000.
[IEE93]	IEEE Society. IEEE Standard for Information Technology - Protocols for Distributed Simulation Applications: Entity information and Interaction. IEEE Standard 1278-1993. IEE Society, New York. 1993.
[Imm02]	Immersion Corporation. *3D Technologies.* Immersion Corporation. 2002. Immersion Corporation. Site: http://www.immersion.com/ 3D Technologies CyberForce Site: http://www.immersion.com/products/3d/interaction/cyberforce.shtml 3D Technologies CyberTouch Site: http://www.immersion.com/products/3d/interaction/cybertouch.shtml
[Jai00]	Jain, Raj. *ATM Name Service.* Department of Computer and Information Science, The Ohio State University. 2000.
[Kar97]	Karr, Clark R; Reece, Douglas; Franceshini, Robert. *Synthetic Soldiers.* IEEE Spectrum, The Institute of Electrical and Electronic Engineers Inc, New York. March 1997.
[Kaz93]	Kazman, R. *Making WAVES: On the Design of Architectures for Low-end Distributed Virtual Environments.* Proceeding of the IEEE Virtual Reality Annual International Symposium. September 1993.
[Kim01]	Kim, Chi Wan; Chee, Yam San; Hooi, Chit Meng. *Supporting User Extensibility in a Networked Virtual Reality Environment.* Proceedings of ICCE/Schoolnet 2001 Conference in Education. 2001.
[Kin98]	Kindratenko, Volodymyr; Kirsch, Berthold. *Sharing Virtual Environments over Transatlantic ATM Network in Support of Distant Collaboration in Vehicle Design.* National Centre for Supercomputing Applications (NCSA), University of Illinois at Urbana-Champaign, USA. March 1998.
[Kru01]	Krus, Mike; Bourdot, Patrick; Guisnel, Francoise; Thibault, Guillaume. *Levels of Detail and Polygonal Simplification.* ACM Crossroads. 24 January 2001. Site: http://www.acm.org/crossroads/xrds3-4/levdet.html Site: http://www.limsi.fr/Individu/krus/LODS/

Key	Reference
[Lam02]	Lambarelli, Livio. *White Paper SHEDS LIGHT ON ATM Service Categories.* The ATM Forum. 3 August 2002. Site: http://www.atmforum.com/pages/library/53bytes/backissues/v4-2/0896-05.html
[Lea96a]	Lea, Rodger; Honda, Yasuaki; Matsuda, Kouichi; Rekimoto, Jun. *Technical Issues in the Design of Scalable Shared Virtual World.* Sony Computer Science Laboratory, Tokyo, Japan. 1996.
[Lea96b]	Lea, Rodger; Honda, Yasuaki; Matsuda, Kouichi. *Virtual Society: Collaboration in 3D spaces on the Internet.* Sony Computer Science Laboratory, Tokyo, Japan. 19 September 1996.
[Lea97]	Lea, Rodger; Raverdy, Honda, Yashuhiko; Matsuda, Kouichi; Matsuda, Satoru. *Community Place: Architecture and Performance.* Sony Computer Science Laboratory, Tokyo, Japan. 26 March 1997.
[Leo92]	Leonard, Brett. *The Lawnmower Man.* New Line Cinema. 1992. ASIN: 6-302-48341-7
[Leo98]	Leow, A.S; Pung, H.K. *A Multipoint-to-multipoint Multicast over ATM Supporting Heterogeneous and Flexible QoS.* School of Computing, National University of Singapore, Kent Ridge Road, Singapore. 1998.
[Li97]	Li, K. Memory Coherence in Shared Virtual Memory Systems. In ACM transactions on Computer System. Volume 7, No 4. 1989.
[Luc02]	Lucas Films Ltd. *Starwars.* Lucas Films Ltd. 2002. Site: http://www.starwars.com
[Mac02]	Macedonia, Michael. *Games Soldiers Play.* IEEE Spectrum. 2 March 2002. Site: http://www.spectrum.ieee.org/WEBONLY/publicfeature/mar02/
[Mac94]	Macedonia, Michael R; Zyda, Michael J; Pratt, David R; Barham, Paul T; Zeswitz, Steven. *A Network Software Architecture for Large Scale Virtual Environments.* Department of Computer Science, Naval Postgraduate School, Monterey, California. 1994.
[Mac95]	Macedonia, M.R. *A Network Software Architecture for Large Scale Virtual Environments.* PhD dissertation, Naval Postgraduate School, Monterey, California. June 1995.
[Mac02]	Macedonia, Michael R. *The Future of Shared VR.* Presentation. 2002.
[Mar02]	Marconi. *Using the Marconi ATM Service Provider with Winsock 2.* Marconi. 2002.
[Mat97a]	Matsubara, Yukihiro; Toihara, Seiji; Tsukinari, Yuichiro; Nagamachi, Mitsuo. *Virtual Learning Environment for Discovery Learning and Its Application on Operator Training.* IEICE Transactions, Volume E80-D, No. 2. February 1997.
[Mat97b]	Matijasevje, Maja. *A Review of Networked Multi-User Virtual Environments.* The Center for Advanced Computer Studies, Virtual Reality and Multimedia Laboratory, The University of South-western Louisiana. 1997.
[Mav02]	Advanced Interfaces Group. *Maverick.* Advanced Interfaces Group, University of Manchester, England. 2002. Site: http://aig.cs.man.ac.uk/maverik/
[Mil95]	Miller, D.C; Thorpe, J.A. *SIMNET: The advent of simulator networking.* IEE Proceedings. August 1995.

Key	Reference
[Min97]	Minoli, D; Schmidt, A. *Client/Server Applications on ATM Network. 1st Edition.* Manning Publications Co. 15 January 1997. ISBN: 1-884-77732-5
[Moc87a]	Mockaptris, P. *RFC1034: Domain Names – Concepts and Facilities.* Network Working Group. November 1987. Site: http://www.faqs.org/rfcs/rfc1034.html
[Moc87b]	Mockaptris, P. *RFC1035: Domain Names – Implementation and Specification.* Network Working Group. November 1987. Site: http://www.faqs.org/rfcs/rfc1035.html
[Mog84]	Mogul, Jeffrey. *Broadcasting Internet Datagrams - RFC 919.* Computer Science Department. Stanford University. October 1984.
[Mor96]	Mori, Kensaku; Urano, Akihiro; Hasegawa, Jun-ichi; Toriwaki, Jun-ichiro; Anno, Hirofumi; Katada, Kazuhiro. *Virtualised Endoscope System - An Application of Virtual Reality Technology to Diagnostic Aid.* IEICE Transactions, Volume E79-D, No 6. June 1996.
[Mor97]	Morse, Katherine L. *Interest Management in Large-Scale Distributed Simulations.* Department of Information and Computer Science, University of California, Irvine. 1997.
[Mun99]	Mundell, Matthew. *Towards a Virtual Operating Environment.* Rhodes University, South Africa. 2 November 1999.
[Nat01]	US National Library of Medicine. *The Visible Human Project.* National Institute of Health, Department of Health and Science. July 2001. Site: http://www.nlm.nih.gov/research/visible/visible_human.html
[NCS95a]	The Board of Trustees – NSCA. *Virtual Reality – History.* The University of Illinois. 1995 Site: http://archive.ncsa.uiuc.edu/Cyberia/VETopLevels/VR.History.html
[Nor02]	Norwood, Carl. *Olympus Eye-Trek FMD-700.* Computer Shopper, Dennis Publishing Ltd, London. February 2002.
[NPS02]	NPSNET. *Shared VR.org.* NPSNET. 2002. Site: http://www.npsnet.nps.navy.mil/
[Oh97]	Oh, Seiwoonf; Kado, Daisuke, Kado; Fujikawa, Kazutoshi; Matsuura, Toshio; Shimojo, Shinji; Arikawa, Masatoshi. *QoS mapping for networked Virtual Reality Systems.* Department of Information and Mathematical Sciences, Graduate School of Engineering Science, Osaka University, Toyonaka, Osaka, Japan; Graduate School of Information Science, Nara Institute of Science and Technology; Osaka City University; Hiroshima City University. 1997.
[Onv95]	Onvural, Raif 0. *Asynchronous Transfer Mode Networks: Performance Issues. Second Edition.* Artech House, London. 1 October 1995. ISBN: 0-890-06804-6
[Par01]	Park, Sungju; Lee, Dongman; Lim, Mingyu; Yu, Chansu. *Scalable Data Management Using User-Based Caching and Prefetching in Distributed Virtual Environments.* Information and Communications University, Korea. 2001

Key	Reference
[Par02]	Paramount Pictures. *Startrek*. Paramount Pictures. 2002 Site: http://www.startrek.com/ Holodeck: http://www.startrek.com/library/techno.asp?ID=105222
[Pet00]	Pettifer, S; Cook, J; Marsh, J; A. West. *Deva3: Architecture for a large-scale virtual reality system.* Proceedings of ACM Symposium in Virtual Reality Software and Technology 2000, ACM Press. October 2000. ISBN 1-58103160-2.
[Pol02]	Polhemus Incorporated. *Polhemus*. Polhemus Incorporated, Colchester. 2002. Site: http://www.polhemus.com
[Pop02]	Popescu, George V; Codella, Christopher F. *An Architecture for QoS Data Replication in Network Virtual Environments.* IBM T.J. Watson Research Center. From the Proceedings of the IEEE Computer Society's Virtual Reality 2002 (VR02). 2002.
[Pos80]	Postel, Jon. *User Datagram Protocol - RFC 768.* Information Sciences Institute. University of Southern California. California. 28 August 1980.
[Pos81a]	Postel, Jon. *Internet Protocol - RFC 791.* Information Sciences Institute. University of Southern California. California. September 1981.
[Pos81b]	Postel, Jon. *Transmission Control Protocol - RFC 793.* Information Sciences Institute. University of Southern California. California. September 1981.
[Pres96]	Preston, Martin; Lever, Paul. *Scene Description Issues for Computer Graphics.* Computer Graphics Unit, Manchester Computing, University of Manchester, Manchester, England. 1996.
[Pun00]	Pung, Hung Keng; Bajrach, Naftali. *A Programmable ATM Multicast Service with Congestion Control.* IEICE Transactions. Volume E83-B, No 2. February 2000.
[Pur00]	Purbrick, Jim; Greenhalgh, Chris. *Extending Locales: Awareness Management in Massive-3.* School of Computer Science and Information Technology, University of Nottingham, UK. 2000.
[Pur99]	Purbrick, Jim. *HIVE Kernel Locale Facilities.* 1999.
[Pyc00]	Pycraft, M; Singh, H; Pjihlela, K. *Operations Management.* Pearson Education, South Africa. 2000. ISBN: 1-868-91070-9
[Qua02]	Id Software. *Quake.* Id Software. 2002. Site: http://www.quake.com
[Qui00]	Quinn, Bob. *A rough Summary of IPv6.* Emailed to the Winsock2 emailing list. 2000.
[Roe97]	Reohl, Bernie. *Channelling the Data Flood.* IEEE Spectrum, The Institute of Electrical and Electronic Engineers Inc, New York. March 1997.
[Roe99]	Reohl, Bernie. *Shared Worlds.* VR News, Volume 8, Issue 2. March 1999. Site: http://www.vrnews.com/issuearchive/vr0802/vrn0802shw.html
[Ror99]	Rorke, Michael; Bangay, Shaun. *A Generic Virtual Reality Interaction System and its Extensions Using the Common Object Request Broker Architecture (CORBA).* Computer Science Department, Rhodes University, Grahamstown, South Africa. Published in SATNAC 1999. 1999.
[Ros01]	Rosen, Eric; Viswanathan, A; Calln, R. *Multi-Protocol Label Switching Architecture - RFC 3031.* The Internet Society. January 2001.

Key	Reference
[Sat00]	Sato, Fumiaki; Minamihata, Kunihiko; Mizuno, Tadanori. *A Totally Ordered and Secure Multicast Protocol for Distributed Virtual Environment.* Faculty of Information, Shizuoka University, Japan. 2000.
[Sat98]	Sato, Fumiaki; Minamihata, Kunihiko; Fukuoka, Hisao; Mizuno, Tadanori. *A Reliable Multicast Framework for Distributed Virtual Reality Environment.* Faculty of Information, Shizuoka University, Japan. 1998.
[Sen97]	Sense8. World2World[tm] Release 1 Technical Overview. Sense8 Corporation, Mill Valley, California. 1997. Site: http://www.sense8.com
[Sen98]	Sense8 Corporation. *WorldToolKit[tm] Release 8 Technical Overview.* Sense8 Corporation, Mill Valley, California. February 1998. Site: http://www.sense8.com
[Sil02]	Silicon Graphics Inc. *SGI Homepage.* Silicon Graphics Inc. 2002. http://www.sgi.com/ Silicon Graphics Inc. *OpenGL: High Performance 2d/3d Graphics.* Silicon Graphics Inc. 2002. Site: http://www.OpenGL.org
[Sin94]	Sing, G; Serra, L; Prg, W; et al. *BrickNet: A Software Toolkit for network-based virtual environments.* PRESENCE: Teleoperators and Virtual Environments. Volume 3, No 1, Page 19-34. 1994.
[Sin99]	Singhal, Sandeep; Zyda, Michael. *Networked Virtual Environments.* ACM Press, SIGGRAPH Series, Addison-Wesley, London. 1999. ISBN: 0-201-32557-8
[Sme01]	Smed, Jouni; Kaukorantu, Timo; Hakonen, Harri. *Aspects of Networking in Multiplayer Computer Games.* Department of Computer Science, University of Turku, Finland. 2001.
[Sno94]	Snowdon, D.N; West, A.J. *AVIARY: Design Issues for Future Large-Scale Virtual Environments.* Presence, Volume 3, No 4. 1994.
[Sno96]	Snowdon, D.N. *AVIARY: A Model for a General Purpose Virtual Environment.* PhD Thesis, The University of Manchester, 1996.
[Sta93]	Stampe, Dave; Roehl, Bernie; Eagan, John. *Virtual Reality Creations.* Waite Group Press, California. 1993. ISBN: 1-878-73939-5
[Sta96]	Stardust Technologies. *Windows Sockets 2 Protocol-Specific Annex, Revision 2.0.3.* Stardust Technologies. 10 May 1996.
[Sta97]	Stardust Technologies. *Windows Sockets 2 Application Programming Interface, Revision 2.2.2.* Stardust Technologies. 7 August 1997.
[Ste93]	Steed, Anthony. *A Survey of Virtual Reality Literature.* Department of Computer Science, Queen Mary and Westfield College, Mile End Road, London. 22 October 1993.
[Str00]	Straaten, P van der. *Interaction Affecting the Sense of Presence in Virtual Reality.* Delft University of Technology. December 2000. Site: http://is.twi.tudelt..nl/~schuemie/vr.html

Key	Reference
[Sun02]	Sun Microsystems. Sun Homepage. Sun Microsystems. 2002. http://www.sun.com/
[Tat98]	Tatipamula, Mallikarjun; Khasnabish, Bhumip. *Multimedia Communications Networks: Technologies and Services*. Artech House, London. 1998. ISBN: 0-890-06936-0
[Ten97]	Tennenhouse, David L; Smith, Jonathan M; Sincoskie, David W; Wetherall, David K; Minden, Gary J. *A Survey of Active Network Research.* Massachusetts Institute of Technology; University of Pennsylvania; Bell Communications Research; University of Kansas. 1997.
[Tor99]	Toriwaki, Jun-ichiro; Mori, Kensaku. *Recent Progress in Medical Image Processing - Virtualized Human Body and Computer-Aided Surgery.* IEICE Transactions, Volume E82-D, No 3. March 1999.
[Tur97]	Turner, Jonathan S. *Extending ATM Networks for Efficient Reliable Multicast.* Department of Computer Science, Washington University. January 1997.
[Uda98]	Udani, Snajay; Smith, Jonathan; Kessler, Drew, G. VENUS: A New Architecture for Scalable Virtual Environments. Universities of Pennsylvania and Lehigh University. 1998.
[Var98]	Varga, Andreas. *PARSEC: Building the Networking Architecture for a Distributed Virtual Universe.* Institute of Computer Graphics, University of Technology, Vienna, Austria. 1998. Site: http://www.parsec.org/
[Vog01]	Vogel, Jurgen; Mauve, Martin. *Consistency Control for Distributed Interactive Media.* University of Mannheim, Germany. 2001.
[Wan97]	Wang, Shengjin; Sato, Makoto; Kawarada, Hiroshi. *Multiresolution Model Constructino from Scattered Range Data by Hierarchical Cube-Based Segemnentation.* IEICE Transactions, Volume E80-D, No 8. August 1997.
[War02]	Warner Bros. Pictures. *Terminator, Terminator 2, Terminator 3.* Warner Bros. Pictures. 2002. Site: http://www.terminator3.com
[Wes92]	West, A.J; Howard, T.L.K; Hubbold, R.J. *AVIARY: A Generic Virtual Reality Interface.* Virtual Reality Systems, International State-of-the-Art Conference, University of London, 20-21 May 1992.
[Wil01]	Wilson, S; Sayers, H; McNeill, M.D.J. *Using CORBA Middleware to Support the Development of Distributed Virtual Environment Applications.* Virtual Environment Applications Group, School of Computing and Mathematical Sciences, Faculty of Informatics, Magee College, University of Ulster, Northern Island. 2001.
[Wlo95]	Wloka, M. *Lag in Multiprocessor Virtual Reality.* Presence, Volume 4, Issue 1. MIT Press. 1995.
[Wri02]	Wright, Bianca. *SA Computer Magazine*; Volume 10; No 1. SABC Publications, Constantia. January 2002.

Referenced papers are available from their authors. If you do not have access to these papers and are unable to locate their original sources, please contact Kevin Dennis as many of the papers have been purchased or otherwise obtained and due to Copyright Laws cannot be legally distributed as part of this dissertation.

Bibliography

Key	Reference
[Abr00]	Abreu, A De; Rodriguez, O. *Analysis of Design of Virtual Reality Applications in the Web: a Case of Study.* Laboratorio de Computacion Grafica, Facultad de Ciencias. Universidad Central de Venezuela. 2000.
[Ade99]	Adelstein, Frank; Golden, Richard III; Schwiebert, Loren. *Building Dynamic Multicast Trees in Mobile Networks.* New Orleans. 1999.
[Baj97]	Bajaj, Chandrajit; Cutchin, Steve. *Web Based Collaboration-Aware Synthetic Environments.* Shastra Lab, Computer Science Department, Purdue University. 1997
[Bak95]	Bakman, Alex. *How to Deliver Client/Server Applications that work.* Manning Publications, Greenwich. 1995.
[Bal93]	Balaguer, Jean-Francis; Gobbetti, Enrico. *Virtuality Builder II: On the Topic of 3D Interaction.* Computer Graphics Laboratory, Swiss Federal Institute of Technology, Lausanne. 1993
[Bau95]	Bauer, F; Varma, A. *Distributed Algorithms for Multicast Path Setup in Data Networks.* Proceedings of Globecom 1995. pp1374-1378. November 1995.
[Ber02]	Bernier, Yahn. *The Half-Life Networked Game Engine.* Valve, LCC. 2000.
[Bie98]	Bierbaum, Allen; Just, Christopher. *Software Tools for Virtual Reality Application Development.* SIGGRAPH '98 Course 14. 1998.
[Bla95]	Black, Uyless. *TCP/IP & Related Protocols, Second Edition.* McGraw-Hill, Singapore. January 1995. ISBN: 0-07005-553-X
[Bow01a]	Bownman, Doug A; Johnson, Donald B; Hodges, Larry F. *Testbed Evaluation of Virtual Environment Interaction Techniques.* Presence, Volume 10, No 1. Massachusetts Institute of Technology. February 2001.
[Bow01b]	Bowman, Doug A; Kruijff, Ernst; LaViola, Joseph J. Jr; Poupyrew, Ivan. *An Introduction to 3-D User Interface Design.* Presence, Volume 10, Now 1, Massachusetts Institute of Technology. February 2001.
[Bow97a]	Bowman, Doug A; Hodges, Larry F. *An Evaluation of Techniques for Grabbing and Manipulating Remote Objects in Immersive Virtual Environments.* Graphics, Visualization and Usability Center, College of Computing, Georgia Institute of Technology. 1997.
[Bow97b]	Bowman, Doug A. *Interaction Techniques for Immersive Virtual Environments: Design, Evaluation, and Application.* Graphics, Visualization, and Usability Center, College of Computing, Georgia Institute of Technology. 1997.
[Bow98]	Bowman, Dough A; Koller, David; Hodges, Larry F. *A Methodology for the Evaluation of Travel Techniques for Immersive Virtual Environments.* College of Computing, Atlanta. 1998.
[Bow99]	Bowman, Dough, A. *Interaction Techniques for Common Tasks in Immersive Virtual Environments.* Graphics, Visualization, and Usability Center, College of Computing, Georgia Institute of Technology. June 1999.

Key	Reference
[Bri97]	Brickenell, Craig. *This is Your Brain on VR.* Wired Digital Inc. 27 June 1997. Site: http://www.wired.com/news/print/0,1294,4670,00.html
[Bro97]	Broll, Wolfgang. *Distributed Virtual Reality for Everyone - a Framework for Networked VT on the Internet.* German National Research Centre for Information Technology, Institute for Applied Information Technology, Sankt Augustine, Germany. IEEE 2000.
[Bro99]	Brooks, Fredrick P, Jr. *What's Real About Virtual Reality.* University of North Carolina at Chapel Hill. 1999.
[Bru97]	Brutzman, Don. *Graphics Internetworking: Bottlenecks and Breakthroughs.* Naval Postgraduate School, Monterey, California. 1997. Site: http://www.stl.nps.navy.mil/~brutzman/vrml/breakthroughs.html
[Cam94]	Campbell, A; Coulson, G; Hutchinson, D. A Quality of Service Architecture. ACM SIGCOMM Computer Communication Review, Vol. 24 No. 2. pp 6-27. April 1994.
[Cap00a]	Capps, Michael et al. Developing Shared Virtual Environments. SIGGRAPH 2000 Tutorial. 2000.
[Cap00b]	Capps, Michael; Watsen, Kent; Zyda, Michale. *Cyberspace and Mock Apple Pie: A Vision of the Future of Graphics and Virtual Environments.* SIGGRAPH 2000. 2000.
[Cap97]	Capps, Michael; Stotts, David. *Research Issues in Developing Networked Virtual Environments.* MIT Laboratory for Computer Science, Cambridge, MA, USA. 1997.
[Chi97]	Chiong, John A. *Internetworking ATM for the Internet and Enterprise Networking.* McGraw-Hill. 17 September 1997. ISBN: 0-07011-941-4
[Cho00]	Choukair, Zied; Retailleau, Damien; Hellstrom, Magnus. *Environment for Performaing Collaborative Distributed Virtual Environments with QoS.* Departement Informatique, Technopole de l'Iroise, France. March 2000.
[Cla96]	Clark, Martin P. *ATM Networks.* Wiley and Teubner, Chichester, England. 15 January 1996. ASIN: 0471967017
[Com02a]	IEEE. *IEEE Computer Society.* IEEE. 2002. Site: http://www.computer.org
[Com02b]	IEEE. *IEEE Communications Society.* IEEE. 2002. Site: http://www.comsoc.org/
[Con97]	Conner, Brook; Cutts, Matt; Fish, Russ; Fuchs, Henry; Holden, Loring; Jacobs, Marco; Loss, Brian; Markosian, Lee; Riesenfeld, Rich; Turk, Gref. *An Immersive Tool for Wide-Area Collaborative Design.* The National Science Foundation Center for Computer Graphics and Scientific Visualisation. Brown University, University of North Carolina at Chapel Hill and the University of Utah. 1997.
[COV97]	The COVEN Project. *Various Documents.* The COVEN Project. 1997.
[Cro01]	Cronin, Eric; Filstrup, Burton; Kurc, Anthony. *A Distributed Multiplayer Game Server System.* Electrical Engineering and Computer Science Department, University of Michigan, Michigan, USA. May 2001.

Key	Reference
[Cru93]	Cruz-Neira, Carolina; Sandin, Daniel J; DeFanti, Thoman A. *Surround-Screen Projection-Based Virtual Reality: The Design and Implementation of the CAVE.* Electronic Visualization Laboratory, The University of Illinois at Chicago, 1993.
[DaS00]	DaSilva, Luiz A. *Pricing for QoS-Enabled Networks: A Survey.* Virginia Polytechnic Institute and State University. USA. IEEE Communications Surveys. 2000.
[Dem99]	Dembovsky, Colin. *VRPhysicsEnvironment - A Framework for Collision Detection and Physical Modelling in a Virtual Environment.* Computer Science Department, Rhodes University, Grahamstown. 1999.
[Den98]	Denby, Bryan. *Technology that Sees Underground.* Nottingham University. Dataweek. 11 April 2000.
[Den00]	Dennis, Kevin S; Doyle, Stuart W. *Investigating the Design of a Distributed Virtual Environment over an ATM Network.* Communications Research Group, University of Cape Town, South Africa. Published in SATNAC 2000, Telkom, South Africa. 2000.
[Den01]	Dennis, Kevin S; Doyle, Stuart W. *Investigating the Design of a Distributed Virtual Environment over an ATM Network.* Communications Research Group, University of Cape Town, South Africa. Published in SATNAC 2001, Telkom, South Africa. 2001.
[Don97]	Dong, Jianning; Niehaus, Douglas; Battou, Abdella; Decina, Basil; Dardy, Henry; Sirkay, Vinai; Edwards, Bill. *Performance Benchmarking of Signalling in ATM Networks.* University of Kansas, Naval Research Laboratory, Fore Systems, Sprint Corporation, Kaman Science Corporation. 1997. Site: http://www.atm.cs.ndsu.nodak.edu/~jdong/seminar/atm97.html
[Dor01]	Dorries, Gundula; Zier, Lothar. *How to Do Hight-Speed Multicasting Right!* GMD - German National Research Center for Information Technology, Institute for Media Communication, Sankt Augustin. 2001.
[Ehm01]	Ehmann, Stephen A; Lin, Ming C. *Accurate and Fast Proximity Queries Between Polyhedra Using Convex Surface Decomposition.* EROGRAPHICS 2001, Volume 20, Number 3. 2001.
[Eng99]	English, Miriam. *Why Virtual Reality?* Miriam English. 1999. Site: http://home.mira.net/~miriam/cyberfemvr2.html
[Fia99]	Fiambolis, Panagiotis; Prokopakis, Georgios. *A Network Design Architecture for Distribution of Generic Scene Graphs.* Naval Postgraduate School. September 1999.

Key	Reference
[Fit00]	Fitch, Crosbie. *Cyberspace in the 21st Century: Mapping the Future of Massive Multiplayer Games.* Gamasutra. 20 January 2000. Site: http://wwww.gamasutra.com/features/20000120/fitch_01.htm Fitch, Crosbie. *Cyberspace in the 21st Century: Cyberspace and Twelve Monkeys.* Gamasutra. 13 March 2000. Site: http://wwww.gamasutra.com/features/20000313/fitch_01.htm Fitch, Crosbie. *Cyberspace in the 21st Century: Foundations.* Gamasutra. 1 December 2000. Site: http://wwww.gamasutra.com/features/20001201/fitch_01.htm Fitch, Crosbie. *Cyberspace in the 21st Century: Foundations II.* Gamasutra. 29 December 2000. Site: http://wwww.gamasutra.com/features/20001229/fitch_01.htm Fitch, Crosbie. *Cyberspace in the 21st Century: Part Five, Scalability with a Big 'S'.* Gamasutra. 26 February 2001. Site: http://wwww.gamasutra.com/features/20010226/fitch_01.htm Fitch, Crosbie. *Cyberspace in the 21st Century: Stability Before Security.* Gamasutra. 26 December 2001. Site: http://wwww.gamasutra.com/features/20011226/fitch_01.htm
[Fle94]	Fleishmann, Albert. *Distributed Systems, Software Design and Implementation.* Springer-Verlag, Berlin. May 1994. ISBN: 0-38757-382-8
[For95]	Forsberg, Andrew; Herndon, Kenneth; Zeleznik, Robert. *Aperture Based Selection for Immersive Virtual Environments.* Brown University, Providence. 1995.
[Fuj99]	Fujikawa, Kazutoshi; Yamauchi, Ryo; Kado, Daisuke. *C^3 : A Scalable Networked Virtual Reality System.* Osaka City University, Osaka, Japan. 1999.
[Gab97]	Gabbard, Joseph L; Hix, Deborah. *A Taxonomy of Usability Characteristics in Virtual Environments.* Department of Computer Science, Virginia Polytechnic Institute and State University, Blacksburg. November 1997.
[Gab99a]	Gabbard, Joseph L; Swatz, Kent; Richey, Kevin; Hix, Deborah. *Usability Evaluation Techniques: A Novel Method for Assessing the Usability of an Immersive Medical Visualisation VE.* Proceedings of VWSUN 99, San Francisco, California. 1999.
[Gab99b]	Gabbard, Joseph L; Hix, Deborah. *Usability Engineering for Virtual Environments through a Framework of Usability Characteristics.* Virginia Tech and Virtual Prototyping and Simulation Technologies Inc, Blacksburg, USA. 1999
[Gor98]	Gordon-Cumming, Ian. *Telepresence - People are the Content.* Intelligence. July 1998.
[Gre97]	Greenhalgh, Chris. *Analysing movement and world transitions in virtual reality tele-conferencing.* Department of Computer Science, University of Nottingham, U.K. September 1997.
[Gre97]	Gregory, Kate. *Special Edition, Using Visual C++ 5.0.* Que Corporation, United States of America. May 1997. ASIN: 0789711451

Key	Reference
[Gre98]	Greenhalgh, Chris. *Predicting Network Traffic for Collaborative Virtual Environments.* School of Computer Science and Information Technology, University of Nottingham, 1998.
[Gre99a]	Greenhalgh, Chris. *HIVEK, The HIVE Project Distributed VR Runtime Kernel.* School of Computer Science and Information Technology, University of Nottingham, 1999.
[Gre99b]	Greenhalgh, Chris. *Realising Flexible Consistency in HIVEK.* School of Computer Science and IT, University of Nottingham, UK. 24 August 1999.
[Gre99c]	Greenhalgh, Chris; Benford, Steve; Craven, Mike. *Patterns of Network and User Activity in an Inhabited Television Event.* School of Computer Science and IT, University of Nottingham, Nottingham, UK. 1999.
[Hai99]	Haider, Ali; Kaneko, Toyohisa. *Automatic Reconstruction of 3D Human Face from CT and Color Photographs.* IEICE Transactions, Volume E82-D, No. 9. September 1999
[Hal96]	Halsall, Fred. *Data Communications, Computer Networks and Open Systems, Forth Edition.* Addison-Wesley Publishing, United States of America. 1996. ISBN: 0-20142-293-X
[Han94]	Handel, R; Huber, M.N; Schrober, S. *ATM Networks - Concepts, Protocols, Applications.* Addison-Wesley. 1994. ISBN: 9-201-42274-3
[Hel91]	R Held; N Durlach. *Chapter 14: Telepresence, time delay and adaptation - Pictorial Communication in Virtual and Real Environments.* Taylor and Francis Ltd. 1991. ISBN: 0-74840-008-7
[Hub01]	Hubbold, R; Cook, J; Keates, M; Gibson, S; Howard, T; Murta, A; West, A; Pettifer, S. *GNU/MAVERIK: A micro-kernel for large-scale virtual environments.* Presence: Teloperators and Virtual Environments. February 2001. ISBN: 1054-7460.
[IEI02]	IEICE. *IEICE Transactions.* IEICE Electronic Publication Committee. 2002. Site: http://search.ieice.org
[Isd98]	Isdale, Jerry. *What is Virtual Reality?* Jerry Isdale. October 1998. Site: http://www.cms.dmu.ac.uk/~cph/VR/whatisvr.html
[Jon98]	Johnson, Andrew; Roussos, Maria; Leigh, Jason; Vasilakis, Christina; Barnes, Craig; Moher, Thomas. Electronic Visualisation Laboratory and Interactive Computing Environments Laboratory, University of Illinois at Chicago, Chicago. 1988.
[Kal98]	Kallman, Marcelo; Thalmann, Daniel. *Modelling Objects for Interaction Tasks.* EPFL Computer Graphics Lab, Lausanne, Switzerland. 1998
[Kam00]	Kamei, Katsuyuki; Huy, Wayne; Tamada, Takashi; Seo, Kazuo. *Modeling of Urban Scenes by Aerial Photographs and Simply Reconstructed Buildings.* IEICE Transactions, Volume E83-D, No. 7. July 2000.
[Kaz92]	Kazman, Rick. *Load Balancing, Latency Management and Separation of Concerns in A Distributed Virtual World.* Department of Computer Science, University of Waterloo, Waterloo, Ontario, Canada. 1992.

Key	Reference
[Kes99]	Kessler, G. Drew. *A Framework for Interactors in Immersive Virtual Environments.* University of Pennsylvania, Philadelphia. 1999.
[Kim98]	Kin, G. Joughyun; Kang, Kyo, Chul; Kim, Hyehung; Lee, Jiyoung. *Software Engineering of Virtual Worlds.* Department of Computer Science and Engineering, Pohang University of Science and Technology, Korea. 1998.
[Kir01]	Kirstein, Mark; Burney, Kneko; Paxton, Mike; Bergstrom, Ernie. *Moving Towards Broadband Ubiquity in U.S. Business Markets.* Cahners In-Stat Group. April 2001.
[Ko97]	Ko, Dong-il; Choi, Yanghee. *Design of Network Protocols for Distributed Virtual Environments.* ETRI, Multimedia Technology Department. 1997.
[Kri96]	Krivda, D.D. *Analysing ATM Adapter Performance. The Real Meaning of Benchmarks.* Efficient Networks. 1996. Site: http://www.efficient.com/dox/Em.ps
[Kri98]	Kritzinger, Pieter. *Introduction to Computer Networks, Edition 6.0.* UCT Press, Cape Town, South Africa. 1998.
[Kuw96]	Kuwahara, Noriaki; Shiwa, Shin-ichi; Kishino, Fumio. *A Method of Displaying Virtual Spaces of Natural Scenes Employing Fractal-Based Shape Data Simplification and Visual Properties.* IEICE Transactions, Volume E79, No 6. June 1996.
[Lea96]	Lea, Rodger; Raverdy, Pierre Guillaume; Honda, Yashuhiko; Matsuda, Kouichi. *Scaling a Shared Virtual Environment.* Sony Computer Science Laboratory, Tokyo, Japan. 16 September 1996.
[Lea97]	Lea, Rodger; Honda, Yasuaki; Matsuda, Kouichi; Hagsand, Olof; Stenius, Martin. *Issues in the Design of Scalable Shared Virtual Environment for the Internet.* Sony Computer Science Laboratory, Tokyo, Japan and the Swedish Institute of Computer Science, Kirsta, Sweden. 1997.
[Lia00]	Liang, Donglin. *A Survey on ATM Security.* Ohio State University. July 2000.
[Lim99]	Lim, Mingyu; Lee, Dongman. *Improving Scalability Using Sub-Regions in Distributed Virtual Environments.* Collaborative Distributed Systems and Networks Laboratory, Information and Communication University, Korea, Published at ICAT'99. 1999.
[Lui98]	Lui, John C.s; Chan, M.F; So, Oldfield K.Y; Tam, T.S. *Balancing Workload and Communication Cost for a Distributed Virtual Environment.* Department of Computer Science and Engineering, The Chinese University of Hong Kong. 1998.
[Lui99]	Lui, John C.S; So, Oldfield, K.Y; Tam, T.S. *Deriving Communication Sub-graph and Optimal Synchronizing Interval for a Distributed Virtual Environment System.* Department of Computer Science and Engineering, The Chinese University of Hong Kong, Hong Kong. 1999.
[Mac96]	MacIntyre, Blair; Feiner, Steven. *Language-Level Support for Exploratory Programming of Distributed Virtual Environments.* Department of Computer Science, Columbia University, New York, New York. November 1996.
[Mac97]	Macedonia, Michael R; Zyda, Michael J. *A Texomony for Networked Virtual Environments.* Fraunhofer Center for Research in Computer Graphics, Providence and the Computer Science Department, Naval Postgraduate School, Monterey California. 1997.

Key	Reference
[Mas98]	Mascarenhas, Rajesh; Karumuri, Dinkar; Buy, Ugo; Kenyon, Robert. *Modelling and Analysis of Virtual Reality Systems with Time Petri Nets.* University of Illinois. Chicago. 1998
[Mas99]	Masunaga, Yoshifumi. *New Generation Database Technologies for Collaborative Work Support and Spatio-Temporal Data Management.* IEICE Transactions, Volume E82-D, Number 1. 1 January 1999.
[Mat00]	Mathy, Laurent; Edwards, Christopher; Hutchinson, David. *The Internet: A Global Telecommunications Solution?* Lancaster University. IEEE July 2000.
[Mau00]	Mauve, M. *Distributed Interactive Media.* Infix, Berlin. 2000. ISBN: 3-89838-471-3
[McD98]	McDaysan, David E; Spohn, Darren L. *Hands-On ATM.* McGraw–Hill, United States of America. January 1998. ISBN: 0-07045-047-1
[Min95a]	Mine, Mark R. *ISSAC: A Virtual Environment Tool for the Interactive Construction of Virtual Worlds.* Department of Computer Science, University of North Carolina. 5 May 1995.
[Min95b]	Mine, Mark R. *Virtual Environment Interaction Techniques.* Department of Computer Science, University of North Carolina. 5 May 1995.
[Min98]	Mine, Mark R; Brooks, Fredrick P. Jr; Sequin, Carlo H. *Moving Objects in Space: Exploiting Proprioceptaion in Virtual-Environment Interaction.* University of North Carolina, Chapel Hill and University of California, Berkeley. 1998.
[Mos96]	Moser, L.E; Narasimhan, P; Melliar-Smith, P.M. *Object-Orientated Programming of Complex Fault-Tolerant Real-Time Systems.* Department of Electrical Engineering, University of California, Santa Barbara. 1996.
[Mul95]	Mullender, Sape. *Distributed Systems, Second Edition.* ACM Press New York, New York. July 1993. ISBN: 0-20162-427-3
[Nae99]	Naemura, Takeshi; Kaneko, Masahide; Harashima, Hiroshi. *Compression and Representation of 3-D Images.* IEICE Transactions, Volume E82-D, No. 3. March 1999
[NEC02]	NEC Research Institute. *ResearchIndex.* NEC Research Institite. 2002. Site: http://citeseer.nj.nec.com
[NSC95a]	NCSA, Board of Trustees of the University of Illinois. *Virtual Reality: VR Interface Devices.* NCSA, Board of Trustees of the University of Illinois. 1995. Site: http://archive.ncsa.uuc.edu/Cyberia/VETopLevels/VR.Interface.html
[Obj98]	Object Management Group Inc. *CORBAservices: Common Object Service Specification.* Object Management Group Inc. December 1998.
[Oli97]	Oliveira, Jauvane C. De; Shirmohammadi, Shervin; Georgannas, Nicholas D. *Collaborative Virtual Environment Standards: A Performance Evaluation.* Multimedia Communications Research Laboratory, School of Information Technology and Engineering, University of Ottawa, Ottawa, Canada. 1997.
[Onv95]	Onvural, Raif O. *Asynchronous Transfer Mode Networks, Performance Issues.* Artech House, London, England. 1995. ASIN: 0890066620

Key	Reference
[Pan98]	Pan, Heng. *SNMP-Based ATM Network Management.* Artech House. London. 1998. ISBN: 0-89006-983-2
[Par94]	Parker, Timothy. *Teach Yourself TCP/IP in 14 Days.* Sams Publishing, United States of America. 1994. ASIN: 0672305496
[Par97]	Park, Kyoung Shin; Kenyon, Robert V. *Effects of Network Characteristics on Human Performance in a Collaborative Virtual Environment.* Electronic Visualisation Laboratory, University of Illinois at Chicago, Chicago. 1997.
[Pet01]	Pettifer, S; Cook, J; Mariani, J. *Towards real-time interactive visualisation virtual environments: A case study of q-space.* Proceedings of the International Conference on Virtual Reality 2001. May 2001. ISBN: 535-3545.
[Pet97a]	Pettifer, S.R; West, A.J. *Deva: A coherent operating environment for large scale virtual reality applications.* Proceedings of Virtual Reality Universe '97. April 1997.
[Pet97b]	Pettifer, S.R; West, A.J. *Worlds within worlds, on the topology of virtual space.* Proceedings of Cyberconf 97. May 1997.
[Pim94]	Pimentel, Ken; Teixeira, Kevin. *Virtual Reality Through the New Looking Glass, Second Edition.* McGraw–Hill, United States of America. November 1994. ISBN: 0-07050-168-8
[Pou98]	Poupyrev, I; Weghorst, S; Billinghurts, M; Ichikawa, T. *Egocentric Object Manipulation in Virtual Environments: Empirical Evaluation of Interaction Techniques.* EROGRAPHICS 98, Volume 17, Number 3. 1998.
[Pry93]	Pryker, Martin de. *Asynchronous Transfer Mode, Solutions for Broadband ISDN, Second Edition.* Ellis Horwoood, West Sussex, England. 1993.
[Rab00]	Rabhi, Fethi A; Cai, Helen; Tompsett, Brian C. *A Skeleton-Based Approach for the Design and Implementation of Distributed Virtual Environments.* School of Information Systems, The University of New South Whales, Australia and Department of Computer Science, The University of Hull, Hull, UK. 2000.
[Ray88]	Raynal, Michel. *Distributed Algorithms and Protocols.* John Wiley & Sons, Chichester, England. February 1988. ASIN: 0471917540
[Rey00]	Reynolds, Craig. *Interaction with Groups of Autonomous Characters.* Research and Development Group, Sony Computer Entertainment America. 2000.
[Roe95]	Roehl, Bernie. *Distributed Virtual Reality -- An Overview.* Bernie Roehl. June 1995. Site: http://sunee.uwaterloo.ca/~broehl/distrib.html
[Roe97]	Roehle, Bernie. *Channelling the Data Flood.* IEEE Spectrum, New York, United States of America. March 1997.
[Ror98]	Rorke, Michael; Bangay, Shaun; Wentworth, Peter. *Virtual Reality Interaction Techniques.* Computer Science Department, Rhodes University, Grahamstown, South Africa. Presented at SATNAC. 1998.
[Ror99a]	Rorke, Michale. *Designing and Implementing a Virtual Reality Interaction Framework.* Computer Science Department, Rhodes University, Grahamstown, South Africa. 1999.

Key	Reference
[Ror99b]	Rorke, Michale; Bangay, Shaun. *The Virtual Remote Control - An Extensible, Virtual Reality, User Interface Device.* Computer Science Department, Rhodes University, Grahamstown, South Africa. 1999.
[Rou99]	Roux, Matthew John. *ATOM: A distributed system for video retrieval via ATM networks.* University of Cape Town, South Africa. 1999.
[Rud98]	Ruddle, Roy A; Payne, Stephen J; Jones, Dylan M. *Navigating Large-Scale "Desktop" Virtual Buildings: Effects of Orientation Aids and Familiarity.* University of Wales, Cardiff. Presence, Volume 7, No 2. Massachusetts Institute of Technology. April 1998.
[San00]	Sandvig, Cary; Schell, Jesse. *Panda 3D.* Disney. 2000.
[San99]	San, Hor Li; Yonekura, Tatsuhiro. *Pseudo-real-time Phenomenon in an Augmented Distributed Virtual Environment (ADVE) with Lag.* Department of Computer and Information Science, Faculty of Engineering, Ibaraki University, Hitachi City, Japan. 1999.
[Sat01]	Sato, Mie; Lakare, Sarang; Wan, Ming; Kaufman, Arie; Liang, Zhengrong; Wax, Mark. *An Automatic Colon Segmentation for 3D Virtual Colonoscopy.* IEICE Transactions, Volume E84-D, No. 1. January 2001.
[Sav94]	Saverino, Michael A. *ATM for Virtual Environments in Advanced Manufacturing Applications.* The ATM Forum. 994. Site: http://www.atmforum.com/pages/library/53bytes/backissues/others/53bytes-0994-9.html
[Sha89]	Shatz, Sol M; Wang, Jia-Ping. *Tutorial: Distributed Software Engineering.* IEEE Computer Society Press, Washington DC. 1989.
[Sha92]	Shaw, Chris; Liang, Jiandong; Green, Mark; Sun, Yunqi. *The Decoupled Simulation Model for Virtual Reality Systems.* Department of Computer Science, University of Alberta, Edmonton, Alberta, Canada. 1992.
[Sla99]	Slater, Mel. *Measuring Presence: A Response to the Witmer and Singer Presence Questionnaire.* 1999.
[Sme02]	Smed, Jouni; Kaukoranta, Timo; Hakonen, Harri. A Review of Networking and Multiplayer Computer Games. Department of Computer Science, University of Turku, Finland. 2002.
[So98]	So, Oldfield, K.Y; Lui, John, C.S. *Partitioning Problem in a Very Large Scale Distributed Virtual Environment.* Department of Computer Science and Engineering, The Chinese University of Hong Kong, Hong Kong. September 1998.
[Ste98]	Steed, Athony; Tromp, Jolanda. *Experiences with the Evaluation of CVE Applications.* Department of Computer Science, University College London and Communications Research Group, University of Nottingham. 1998.
[Ste94]	Stevens, Marc P; Zeleznik, Robert C; Hughes, John F. *An Architecture for an Extensible 3D Interface Toolkit.* Department of Computer Science, Brown University, Providence. 21 April 1994.
[Ste97]	Stephenson, G; Frampton, R. *Online Virtual Reality: Stretching the Technologies. Online '97 Conference, London, UK.* 9 December 1997.
[Sty97]	Stytz, Martin R; Adams, Tarry; Garcia, Brian; Sheasby, Steven M; Zurita, Brian. *Rapid Prototyping for Distributed Virtual Environments.* Air Force Institute of Technology. IEEE Software. 1997.

Key	Reference
[Sug97]	Sugano, Hiroyasu; Otani, Koji; Ueda, Haruyasu; Hiraiwa, Shinichi; Endo, Susumu; Kohda, Youji. *SpaceFusion: A Multi-Server Architecture for Shared Virtual Environments.* Fujitsu Laboratories Ltd. Published in VRML 97. 1997.
[Tac98]	Tacic, Ivam; Fujimoto, Richard M. *Synchronized Data Distribution Management in Distributed Simulations.* College of Computing, Georgia Institute of Technology, Atlanta. 1998.
[Tai96]	Tai, Charlie. *ATM Specific Extensions.* Intel Architecture Labs. 30 August 1996.
[Tan01]	Tanner, John C. *ATM Lives?* America's Network. 2001. Site: http://www.americasnetwork.com/issues/2001issues/20010501/20010501_backbone.htm
[Tan96]	Tanenbaum, Andrew S. *Computer Networks, Third Edition.* Prentice Hall, New Jersey, United States of America. 6 March 1996 ISBN: 0-13349-945-6
[Ter00]	Terraplay Systems. *The Terraplay System.* Terraplay Systems AB. 2000 Site: http://www.terraplay.com
[Tro97]	Tromp, Jolanda G. *Methodology for Distributed Usability Evaluation in Collaborative Virtual Environments.* Communications Research Group, Department of Computer Science, University of Nottingham, University Park, Nottingham, U.K. 1997.
[UNO97]	University of New Orleans. *Distributed Multicast Tree Generation with Dynamic Group Membership.* University of New Orleans, Computer Science Technical Report. UNOLS – TR97 – 01
[VRM97]	VRML Standards: Site: http://www.web3d.org/Specifications/VRML97/ Site: http://www.vrml.org/
[Wan94]	Wang, Qunjie. *Networked Virtual Reality.* Department of Computer Science, University of Alberta, Edmonton, Alberta. 1994.
[Wei94]	Weiss, Sonia. *Virtual Reality.* Jones International and Jones Digital Century. 1994. Site: http://www.digitalcentury.com/encylco/update/vr.html
[Wei96]	Weikle, Bobbie. *Riding the Perfect Wave: Putting Virtual Reality to Work with Disabilities.* Ball State University. 1996. Site: http://www.pappanikou.uconn.edu/weikle.html
[Win02]	WinSock 2 Mailing List Site: http://www.sockets.com/winsock2.htm
[Wlo95]	Wloka, Matthias, M. *Interacting with Virtual Reality.* Science and Technology Center for Computer Graphics and Scientific Visualization, Brown University, Providence. 1995
[Yan00]	Yang, Jeonghwa; Lee, Dongman. *Scalable Prediction Based Concurrency Control for Distributed Virtual Environments.* Collaborative Distributed Systems and Networks Laboratory, Information and Communications University, Korea. 2000.
[Zel93]	Zeleznik, Robert C; Herndon, Kenneth P; Robbins, Daniel C; Huang, Nate; Meyer, Tom; Parker, Noah; Hughes, John F. *An Interactive 3D Toolkit for Constructing 3D Widgets.* Department of Computer Science, Brown University, Providence. 1993.

Key	Reference
[Zho00]	Zhou, Yi; Murata, Tadao; Defanti, Thomas; Zhang, Hui. *Fuzzy-Timing Petri Net Modelling and Simulation of a Networked Virtual Environment: NICE.* IEICE Transactions, Volume E83-A, Number 11. November 2000.
[Zho99]	Zhou, Y; Murata, Y; DeFanti, T. *Modelling and Analysis of Collaborative Virtual Environments by Using Extended Fuzzy-Timing Petri Nets.* Department of Electrical Engineering and Computer Science, University of Illinois at Chicago. 1999.

Appendix A: Source Code

The source code for the various environment servers (Name Server, Broker Server, Model and Media Server, Control Server and Broadcast Server) and the client simulation software (Tester) can be found on the attached CD. Please note that neither the servers nor the client software will function without ATM hardware and Winsock2 installed.

Appendix B: Architectures

This appendix focuses on the various communication topologies that can be used when creating a distributed system. Certain advantages and disadvantages specific to DVEs are noted to highlight the various architectures' usefulness when considering the creation of a DVE.

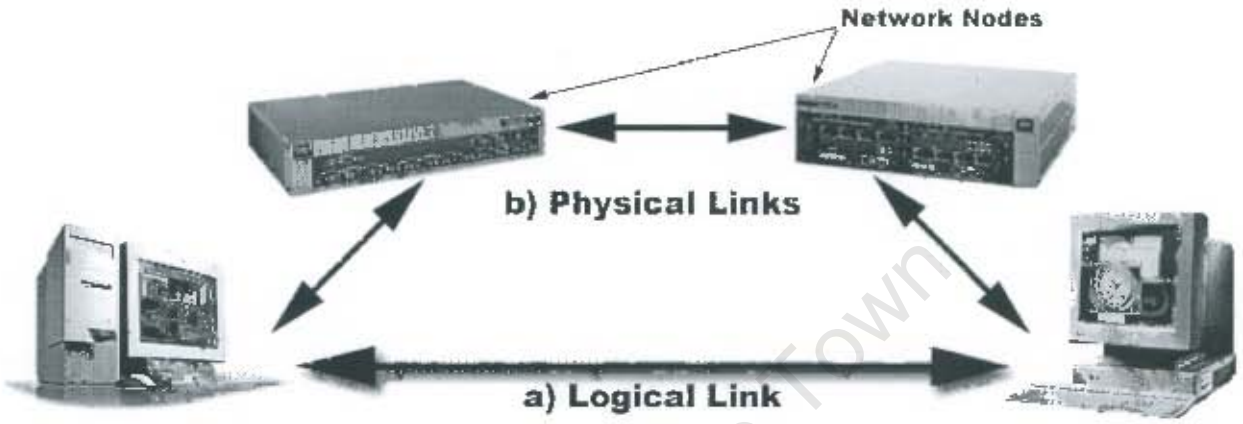


Figure B.1: (a) Logical and (b) Physical Connections

Figure B.1 shows two ways for end terminals to be connected. Figure B.1a shows the *logical connection* between two end-terminals across which information can be sent. Figure B.1b shows one of many possible *physical connections* consisting of the connecting media and networking components. The physical link is often abstracted to allow any application to transparently use the logical link, between two end terminals, as if the two terminals where directly connected.

Often architectures are discussed in terms of the logical connections between terminals in the VE, as this is how information flows, but it is imperative that the limitations of the physical connections are also considered, as these limitations can drastically affect the ideal logical connection's performance. Similarly, the protocols used over the physical link and the information sent over the logical link can effect the link's performance.

The following sections present topologies that can be used to interconnect components of a VE.

B.1 Client-Server Architecture

In the *client-server* communications architecture, terminals send messages via a centralised server. This architecture is shown in Figure B.2.

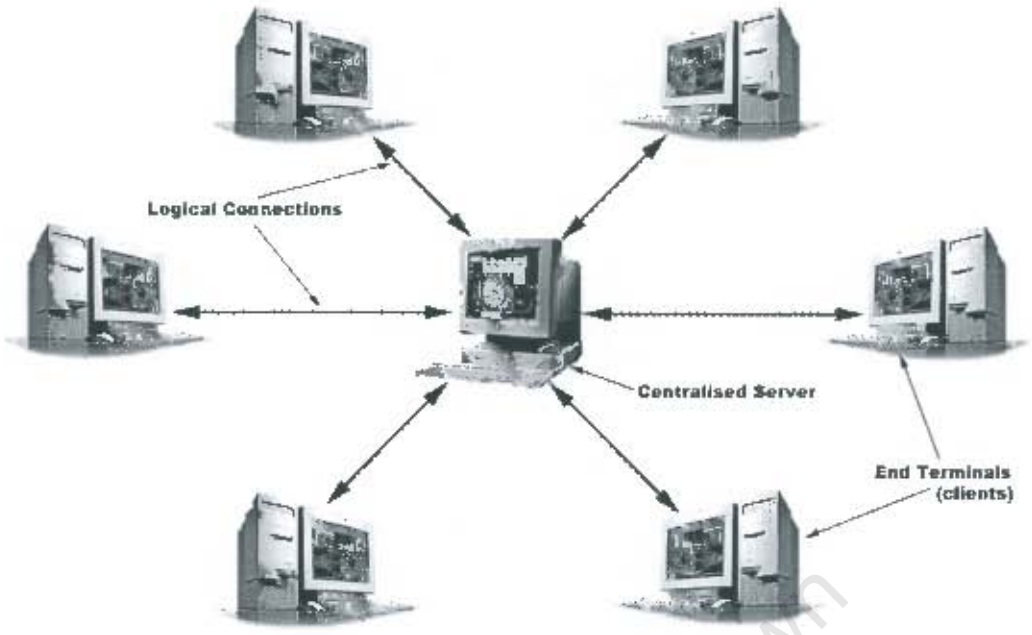


Figure B.2: Logical Star Architecture

This logical architecture, while relatively simple, can be rather convenient as it has many advantages:

- The centralised nature of the architecture allows the server to process any information before passing them to the clients. This is often essential to VE designs.

- A single server is easily able to maintain a database that is completely consistent for all the clients in the VE, allowing joining users to retrieve a view of the active VE that is consistent with the existing users.

- Similarly, since a single server has sequential access to the VE database, any requests are dealt with on a first-come-first-served basis, preventing any inconsistencies or contention issues.

- Another major advantage is that the server can reduce the quantity of information that is sent to an individual client by using various filtering techniques. The client will generally only be interested in a small portion of the VE and any actions that occur outside of this *area of*

interest[12] are not important to the user, and if the server sends this information, it will be a waste of bandwidth and computational resources for both the client and server.

- With prior knowledge of the client's minimum and maximum needs and permissions, the server can filter the information sent to each client. This area of interest for each user can also be dynamically altered depending on the client or server's current load.

- To support an extensive client base and to facilitate growth over a long term of operation, the VE can provide backward compatibility for legacy protocols. As each client receives unique communications from the server, the server needs simply to keep track of which protocol it is using to communicate with the client and to translate any information into the appropriate protocol format before sending to that specific client.

- If the VE is to become commercially viable, there needs to be some way to keep statistics on how the clients' use the VE and to bill them based on this. Keeping these statistics is possible in client-server architectures as all information passes through the central server, allowing detailed statistics of each user's activities to be maintained.

- In order to prevent duplicate user connections and to facilitate the collection of statistics, the server keeps a database of the clients. If security is required for the VE, then passwords and other security features can be added. A single server ensures that no more than one client can participate in the VE with the same user name, as logins are also handled on a first-come-first-served basis.

- Another factor that can easily be overlooked is the fact that the server is one logical entity. The server only has one identifying name or address and this is far easier for a user to remember and far simpler for a client to connect to.

While there are clearly many advantages to this architecture, there are also several disadvantages.

---

[12] The area of interest is the area around the user's point of reference within the environment from which the user wishes to receive information. Often the user will have different visual and auditory areas of interest and the area of interest is also often known as an *aura*.

- The first and most critical disadvantage is that the server is a central point of failure. Should the server in any way fail, the entire VE will cease to function. This is clearly highly undesirable as reliability is an extremely important factor.

- The server has finite resources, both in processing time and incoming and outgoing bandwidth, and results in the limitation of the number of messages that it can process within a given period. This processing can take the form of application of updates to the database, user or server specific filtering, translation of data, billing, security checks or statistical analysis. As the number of clients, and hence messages, increases, the server will reach a point where it can no longer forward information to all connected clients in such a way as to ensure the response times needed for realistic interaction.

When considering the physical links in the architecture, there are several further advantages and disadvantages to this architecture.

- For networks that provide QoS support and where server-side filtering is used, it is possible to map the client's requirements and area of interest to a traffic contract that would be created during connection set-up. By reserving network resources, there is a much lower likelihood of congestion, packet loss, and a deterioration of the client's VE experience. Note that while the bandwidth between the client and the server is reserved, the server may still further limit the quantity of information sent to the client during load conditions. The pre-reservation simply sets an upper limit to the information flow.

- If server-side filtering is not used, all the information will be sent to every client. If the VE is then limited to a LAN that supports a broadcast or multicast mechanism then the server can take advantage of this in order to simultaneously send a single update to all clients. Similarly, should the VE be extended to a WAN, multicasting could be used to connect the server to all the clients.

In contrast to the logical diagram that shows each client having a completely independent connection to the central server, all the logical connection must travel to the server over a single physical connection. The implications of this are two fold. Firstly, the server will be separated from the network should the link between the server and the network in any way break and the entire VE will

cease to function. Secondly, if any of the logical connections "misbehaves" by, for example, using too much bandwidth, then the service offered to any of the other logical connections could be affected. This is shown in Figure B.3.

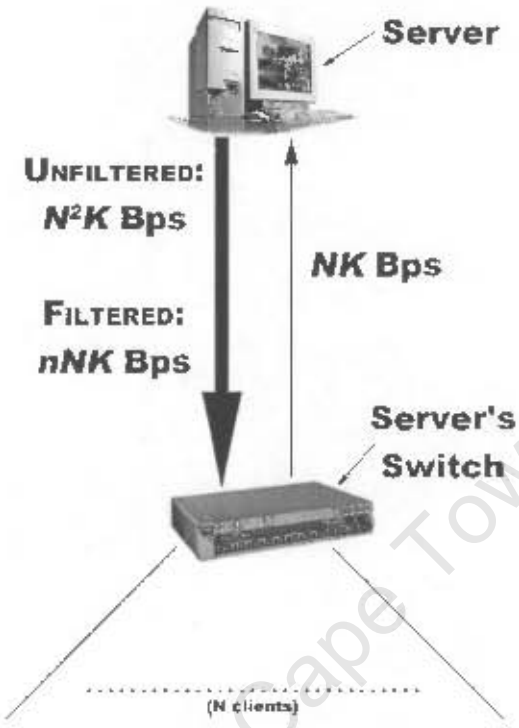


Figure B.3: Limited number of connections between the server and network

- The server does not have an infinite amount of bandwidth on the physical link connecting it to the network. Since all the logical connections share this link, as the number of user increases, so does the bandwidth needed. Eventually, a point will be reached where there is no available bandwidth to accommodate new users or if new users are added, congestion will occur. This severally limits the VE, as scalability is a major concern. If a server has N connected clients who each sending K Bytes per second (B/s), then the data received by the server will be NK B/s. For a server that does not use filtering, the outgoing bandwidth will be N^2K Bps. This is an $O(N^2)$ bandwidth problem. For a server that uses filtering and only sends the data of n users to each client, then the outgoing bandwidth used will be nNK B/s. If N is sufficiently larger than n, then the problem is simply an O(N) bandwidth problem and significantly more efficient.

- If all the clients are forced to connect to a single server, there is no guarantee that the server will be situated in the physical centre of the area covered by the connected clients. This could result in long time delays for messages that have to travel far to reach the server.

B.2 Peer-to-Peer Architecture

From a scalability point of view, the main flaw with the client-server architecture is that the centralised server is the major bottleneck. The servers, no matter how numerous, powerful or expensive, will eventually limit any architecture that is primarily based on this paradigm.

An alternative to the client-server architecture is known as *peer-to-peer*. This architecture removes centralised servers and allows clients to communicate directly with each other in order to share information. This has several advantages.

- One of the main advantages of using a peer-to-peer architecture is that there is no centralised server and thus no centralised point of failure. This is extremely useful when reliability is a concern, as an active peer does not have any other requirements for communication other than that the destination peer is also be active and reachable. The only clients that are directly affected by a peer failing are those that are currently communicating with it or those that need information from it. In these cases, only minor inconsistency should occur and these can sometimes be alleviated using various algorithms. In the worst case, the data needed is no longer available to the environment although this situation should be avoidable if good data distribution algorithms are used. This is in contrast to a client-server architecture, where the entire VE ceases to operate when the server fails or needs to be turned off for an upgrade.

- In a situation where each client only connects to one of its peers when data transfer is required, there is a considerable advantage over the client-server architecture. In such a peer-to-peer environment, the clients are not as heavily loaded as the centralised server could be in a client-server architecture. Each peer only has to be able to handle a relatively small number of connections compared to the total number of users in the VE, and only has to process requests from these connections. The server, however, needs to store a connection for every user in the environment and needs to process any data on these connections. This severely limits the server's capacity to identify unique connections and to process the information from these connections. Using a peer-to-peer architecture, it may be possible to increase the number of clients by a significant number, and under ideal circumstances, the number of clients that can exist in such a VE could be limitless.

- Another benefit of using direct connections is that there is much less latency associated with the transfer of information. In a client-server system, the message has to be passed up to the

server, be processed and then be passed back down to the destination. The use of direct connections removes this extra network latency and server processing time. Interactive VEs have very strict latency requirements, and any architecture that can minimise latency is highly desirable.

Although there are definitely significant advantages to using peer-to-peer, especially in terms of scalability, there are also various disadvantages, listed here.

- If the clients use a fully replicated VE database to ensure consistency then the architecture's benefits are completely removed. To ensure consistency, every client must receive every update and store the entire environment. This is extremely wasteful of bandwidth and space despite the reduction in latency gained in the sending of the updates. The client's network connection will need to be able to receive N^2K B/s, as in the client-server architecture when retransmitting to all clients. However, the client will also have to render the user's view of the VE while trying to processes the incoming updates. In addition, since the target client base is large and consists mainly of members of the public, it is most likely that they will have dial-up lines ranging between 56Kbits/s (modems) and 8Mbits/s (ADSL). Both of these capacities will be used up very quickly as the client base increases.

- If a fully replicated database is not used, algorithms can be used to distribute the database, reducing the amount of information transferred. However, these algorithms are often complex, requiring additional processing time, and ultimately cannot prevent a loss of consistency within the VE should there not be enough resources available, or a significant portion of the DVE fails.

- Since clients in a peer-to-peer architecture typically send the same information to all the clients they are connected to, a network that supports multicasting technology will be very beneficial, as only a single packet would have to be transmitted. The network takes care of the delivery to the multiple receivers. However, the support of multicast in current networks is very limited. In the Internet, there are very few routers that support multicast, and in ATM, most switches cannot handle more than 4096 multicast connections per input port, even in large backbone switches. These restrictions make it virtually impossible to use multicast extensively in any design aimed at a large client base primarily made up of members of the public. In the absence

of multicast, multiple point-to-point connections need to be made. In this case, there will be a large amount of duplicated data being sent on each connection, severely limiting the effective capacity of the physical link. As an example, consider a user using a modem running at 56Kbits/s and generating a 30 Byte packet 30 times per second that needs to be sent to 10 other clients without multicast. That is a total of 72Kbits/s of data, excluding any packet headers. This is clearly well beyond the capability of such a modem connection and any schemes to reduce the rate of sending would probably increase the packet size. The Peer-to-peer architecture has to be used very carefully in a design aimed at the public.

B.3 Hierarchical Tree Architecture

This architecture uses a tree structure to combine elements of the VE. The elements need not specifically be servers or clients although typically the clients are the leaves while each branching node is a server. Each element will connect, to at most, one element at a higher hierarchical level and have any number of elements, from a lower level, connecting to itself. An example of a hierarchical tree is shown in Figure B.4.

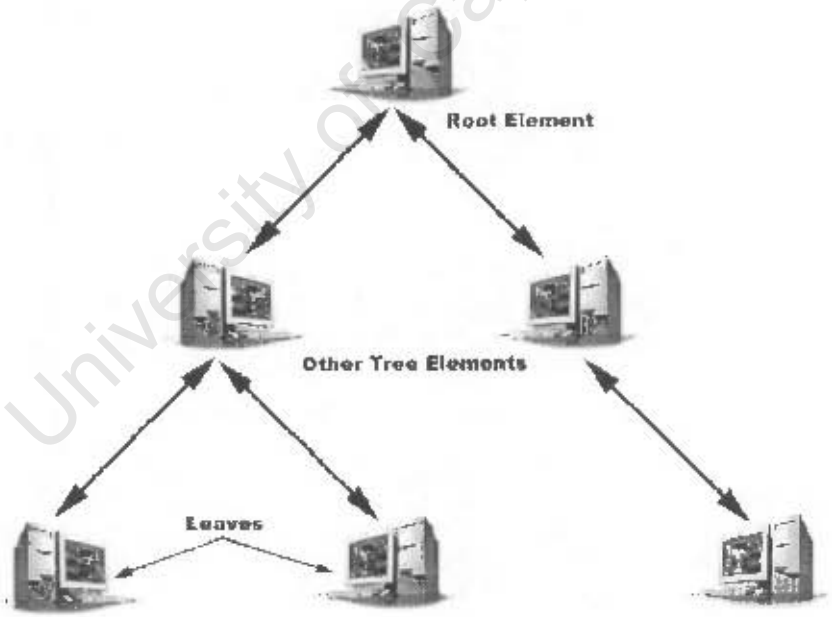


Figure B.4: Hierarchical Tree Architecture

As with any architecture, there can be various advantages to using it although the advantages are particularly application specific. The choice of a hierarchical architecture can be advantageous to an application should it meet the following criteria.

- The most appropriate form of communication between hierarchical levels is where data can be simplified or abstracted before being passed to the higher level using a simple command. If this is not possible, then the elements higher up in the tree will tend to accumulate large quantities of complex data. If the data can be abstracted, then the higher-level elements are able to use this data to make generalised decisions on what needs to be done. The higher-level elements then pass generalised commands back down the tree in order to accomplish the necessary tasks. The abstraction used at each element allows complex results to be achieved using relatively simple commands and limits the quantity of data that has to be sent to the higher-level elements. This is particularly useful when considering address resolution and management schemes although not as useful when considering environment updates. These management schemes can include synchronisation strategies. Since the tree structure is hierarchical and lends itself to abstraction, it might be easier to implement partial replication schemes that can produce better database consistency than those used in peer-to-peer, but not as good as those found in the client-server model.

- The hierarchical tree requires that each element still function properly despite not being able to reach its parent. If the parent is unreachable then only the data stored in the top-most reachable element will be available to the elements directly beneath it. This allows a robustness that permits continued operation within the newly formed tree fragments, although with a reduced service or knowledge base.

- If the hierarchical tree is being used to distribute data, then each element will need to communicate with some or all the other elements in the tree. If this is the case, large amounts of data will travel through the elements in the higher levels, placing an increasing load on each element as the data moves up the tree. Although this may not provide as distributed a solution as a peer-to-peer architecture, it provides a significant improvement over a client-server architecture, as data destined for an element in the same sub-tree will not leave that sub-tree. This reduces the load on each element such that, in the worst case, no element experiences the same maximum N^2KBits/s load as would occur in the centralised server architecture.

However, if the application is delay sensitive as a VE is, then this architecture may be unsuitable as delays can be lengthy and highly variable between users in the hierarchy. Figure B.5 illustrates this.

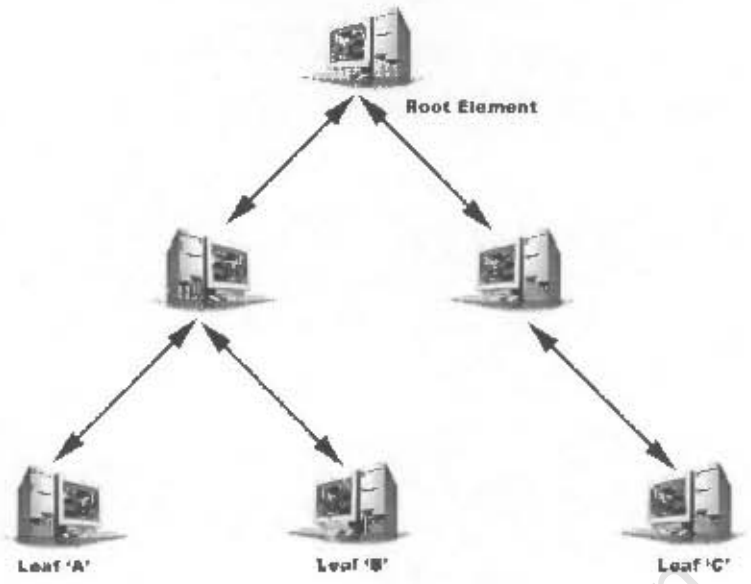


Figure B.5: Hierarchical Tree Architecture with Delays

The data from user "A" will only have to travel via one element in the tree to reach user "B." If the same data were to be sent to user "C" then the path would be far longer, consisting of three elements. As the tree becomes larger, the number of elements traversed, and therefore path-length travelled, increases. This is particularly evident in the worst-case scenario that occurs between elements at the bottom of two different branches of the root element. These variable delays, if not kept within acceptable limits, could easily cause inconsistencies between users in a group within the VE.

B.4 Hybrid Architecture

A *hybrid* architecture combines various aspects of the common architectures to form one more suitable to the needs of an application. The hybrid architecture for HEAVEn is discussed in Chapter 4.

Appendix C: Networking Technologies

Communication has always been significant in humankind's history allowing us to work together and accomplish complex tasks that would not have been possible individually. Computers working together can similarly accomplish tasks that are more complex than they could accomplish individually by sharing such resources as storage space, processing time, I/O devices and computational results. The collaboration resulted in the formation of the first computer networks.

C.1 Characteristics of Network Technologies

Much like languages, programmers developed communication techniques specifically suited to their needs. It is therefore important to ensure that, whether an application is being written for a particular type of computer network or a network is being chosen to support a particular application, the characteristics of both the application and the network are taken into account. This is to ensure that they work together correctly and produce a functional product. The following sections give a brief summary of several of the characteristics by which a technology is measured and a brief summary of several of the more common networking technologies.

C.1.1 Latency

Latency is the delay between one system sending a bit[13] or packet of information and the other receiving it. When applications use only the resources of a single computer, the transfer delay is often minimal but in the case of computer networks, the delays can be relatively long.

The latency of a network directly affects the responsiveness of any network application. If the latency between one computer and another is n seconds, then the round trip response time will be $2n$ seconds, assuming there is almost no delay before the receiver sends the response. There are many sources of latency within a network, for example: propagation delays as the data travels across the media, congestion in the transport media, and delays at both the end terminals and at the network nodes along the path.

[13] A byte consists of 8 bits.

C.1.2 Bandwidth

The network bandwidth is a per unit time measure of the quantity of data that can be sent or received. The data-rate is most significantly affected by the type of cabling and by the node hardware. Some examples are listed in Table C.1.

Means	Description
Modem	Modems are used to transmit data over conventional phone lines and can typically handle rates between 14Kbits/s and 56Kbits/s. Most modems are used to connect home-users to a local dial-up internet service provider.
Ethernet	Ethernet is a networking technology used in LANs. Conventional Ethernet runs at 10Mbits/s and a more modern variation at 100Mbits/s.
ATM	Most ATM implementations are rated at 155Mbits/s although core networks using ATM run at speeds of 10GBits/s or faster although the protocols can still run at minimal speeds of 25Mbits/s.

Table C.1: Bandwidths

Network bandwidth and latency are often confused. Bandwidth is the speed at which data can be added to or removed from a network connection. Latency is the time taken for data to travel from the source to the destination.

C.1.3 Reliability

Reliability is an important factor in a computer network and is a measure of how likely an error will occur. This factor is not easily accounted for due to the myriad of possible ways that errors can occur to prevent data from arriving correctly at its destination. The errors can either be a complete loss of data or an unrecoverable corruption of the data and can be caused in a variety of ways.

Error Cause	Description
Hardware Failure	This is generally the least likely cause of failure, although it does occur and in such cases as cable theft, can have a significant impact on the network. Hardware failure tends to cause a complete loss of data rather than a corruption of the data. The only way to prevent this is to use multiple systems in parallel.

Error Cause	Description
Interference	Interference occurs when communication is interrupted due to interference on the transfer media caused by other unshielded connections or by a 'noisy' environment. The interference causes a corruption of the data and the data may become unrecoverable. In order to protect against the corruption, networking protocols and applications often implement systems to ensure that errors in the data arriving at the destination are detectable or correctable. The most common way of doing this is to add extra information that can be used to check and/or correct the data. The extra bits of information are known as Cyclic Redundancy Check (CRC) bits.
Congestion	Congestion can occur at the various nodes within the network. This is due to the nodes' capacity not being equal to the quantity of data that the nodes need to handle. The capacity is seen in terms of the memory used to store information that needs to be sent. If the queues become full, data to be added to the queues will be lost.

Table C.2: Causes of Network Errors

C.1.4 Network Protocols

For communication, two end systems must be using the same networking protocols or there must be a translation node between them. There are three main aspects of a protocol:

Aspect	Description
Packet Format	Both the end stations need to know the exact structure of the data being transferred. Without this being defined, the receiver will not know what type of packet is being received or where in the packet the data can be found.
Packet Semantics	Appropriate responses to specific packets are important, as computers are not very flexible in their understanding of the data being received.

Aspect	Description
Error Handling	Should errors occur, there needs to be a response by the end-stations. If the protocol is not designed to take into account the possibility that an error could occur, then when one does, the protocol would be left in an incorrect state, causing an error. The strategy for handling an error could be as simple as closing the connection or resetting it and starting again. The error handling rules govern what to do in both the event of receiving an incorrect packet, as well as not receiving a packet at all.

Table C.3: Aspects of Protocol

There are many network protocols offering different services tailored for specific tasks or environments. Since there are so many components used to support the functionality of higher-level applications, there are often many protocols being used at once to perform a single task. This is one of the reason for delays at the end-stations and a cause for latency in communications. As the original data moves through the computer system, into the network, and back out, each module it travels through adds extra information specific to the protocol it is using. This extra protocol information builds up at each level with two effects. The first is that it is impossible for an application to transfer data at the full rate specified for the link, i.e. a 10Mbps Ethernet link will not let a person transfer at 10Mbps. The second is that if there is an error at any point, the data for that level of protocol becomes invalid and this affects the protocols at higher levels too. This is the reason for using CRC checks as they ensure that the data is transferred correctly.

C.2 Some Common Network Technologies

While there are many network technologies available, a few of the most common network technologies currently in use will be focused on here. These sections will not go into detail about the actual workings of the various protocols, but will just give an overview of their structure as well as the services they offer.

C.2.1 Ethernet Networks

Ethernet [IEEE00] is a physical network protocol commonly used in Local Area Networks. This means that it defines how data bits are transferred over a physical cable; in Ethernet's case, this is over wires using electrical signals.

C.2.1.1 Ethernet Basics

Ethernet was originally designed to use a single wire for multiple computers to communicate across, thereby eliminating complex switching equipment and extra wiring that had been in use at the time. Simply being connected to this shared wire was enough to enable an end-station to send or receive data from any other end-station on the network. Due to the way it works, Ethernet is only suitable for LANs and is limited to 10Mbps.

The protocol works by trying to send data out onto the wire. If it detects that some other computer is trying to send data on the common wire at the same time, then it assumes the data sent was made invalid and waits a short random amount of time before trying to send again. Likewise, since the other computer would have also detected another computer trying to send data, it would also wait a random amount of time before trying to resend.

If there are a number of people on the network at the same time and all are trying to transfer large amounts of data, then the reliability of the network is going to drop drastically. If a computer sends and detects a collision because someone else is also trying to send, then it is going to wait a short while. However, when it tries to send again, because of the number of people trying, chances are high that there will be another collision and therefore wait. This might go on for a while before a gap occurs in which the computer can transmit. Due to the number of collisions, the effective capacity of the network is significantly reduced during the congested period.

The second problem with the original Ethernet is that it requires each end of the wire to be properly terminated. If a break occurs at any point along the shared wire then the network cannot function.

With this in mind, a unit called a Hub was developed. This unit required a connection to it per computer. Although this increased the amount of wiring needed, it had advantages in terms of reliability. If a computer sent data, the hub would then copy it out to all the other connected computers. If a computer failed, or its cable got disconnected or damaged, only that computer got cut off from the network, rather than causing the entire network to fail.

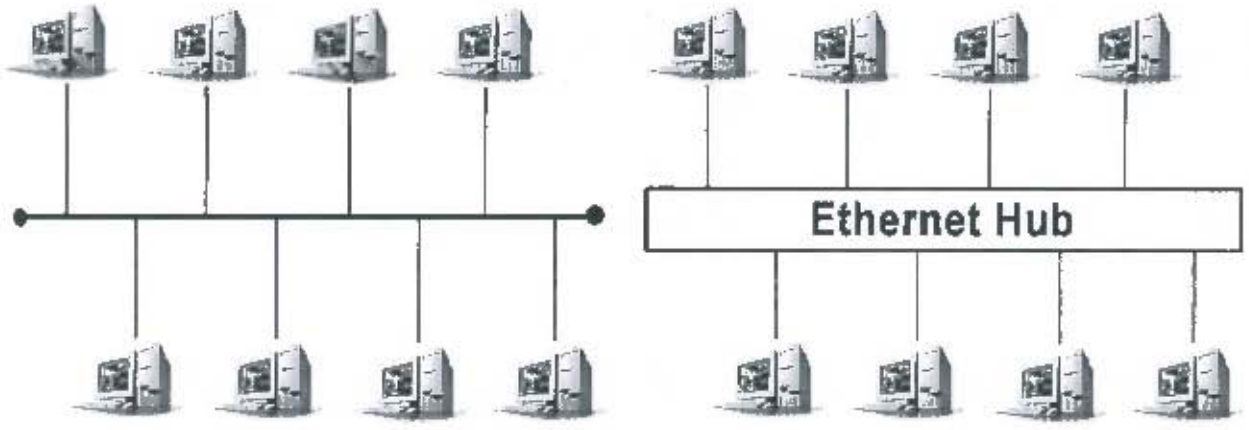


Figure C.1: (a) Ethernet Using a Common Wire (b) Ethernet Using a Hub

The new architecture increased Ethernet's reliability and from the hub was developed the Ethernet Switch. Every computer on an Ethernet network is given a unique identifier known as a Media Access Control (MAC) address. The MAC address is used by other computers to identify what computer data is destined for, and similarly to recognise what data is to be received. The design of the switch made use of these addresses to selectively send data only to where it had to go to. As computers connect to the switch it identifies their MAC addresses and can thus forward appropriate information to a specific destination. This selective forwarding partially solved the problems that arose when everyone tried sending or requesting data at the same time.

The Ethernet Switch, by isolating each computer and intelligently forwarding data, set the stage for future developments in Ethernet technology allowing it to reach speeds of 100Mbits/s (Fast Ethernet) and even 1000Mbits/s (Gigabit Ethernet). Of these two improvements, Fast Ethernet is generally used in conjunction with 10Mbits/s Ethernet for LANs, while Gigabit Ethernet is often used to join LANs together to form a larger network, such as a Metropolitan Area Network (MAN).

C.2.1.2 Ethernet Characteristics

Ethernet is one of the most common LAN technologies in use and this is due to several factors. Since it was one of the early networking protocols, it has been around for some time. It is also relatively easy to use and install. While there were other emerging network technologies at the time, often with advantages, smaller networks chose Ethernet as it was simple and worked "well-enough" for its cost. As people moved on, they tended to stick with what they knew and therefore continued to use Ethernet, thus ensuring its longevity.

From an application programmer's point of view, there are two major characteristics of Ethernet that need to be taken into account. The first is that the original Ethernet is classed as a broadcast network. This is because data sent from one computer can be received by all other computer on the same network, regardless of whether the data was destined for that computer or not. In addition to this, a special MAC address was defined which indicated to the computers on the network that the data was meant for all of them. Later, when Ethernet switches were created, data sent to this MAC address was copied to all the outgoing connections on the switch. This continued to allow broadcast communication, even on the selective forwarding switches.

The second characteristic is that Ethernet does not offer any form of guaranteed Quality of Service (QoS). Ethernet cannot guarantee that the data sent will reach the destination within a specific amount of time. It also cannot guarantee that a connection will have the same quantity of bandwidth to a destination machine for the entire connection's lifespan. If another end-station wishes to send to a destination already receiving data, the switch will forward the data on despite any detrimental effects it might have.

C.2.2 Internet Protocol

One of the major problems with having a variety of physical networks is that it is very unlikely that there will be a single physical network connecting any two points long distances apart. This means that data transfers between arbitrary computers is often not possible, as the data will have to traverse different physical networks, each likely to be using a different physical network technology.

The design of the Internet Protocol [Pos81a] solved this problem. The protocol defines communication on a logical network that is independent of the hardware used to send data from one point to another. It can therefore provide a standardised mechanism for computers on different physical networks to communicate with each other. The Internet Protocol (IP) is now the predominant network protocol used to transfer data between computers on the variety of networks that make up the Internet.

C.2.2.1 A Brief History

The Internet Protocol is one protocol in a suite of protocols developed under the Defence Advanced Research Projects Agency (DARPA) of the United States of America. The other protocols in the suite operate with IP to provide extended functionality for specific purposes.

There were two main driving forces behind the initial development of this suite of protocols, both military related. The first was so that the military could have a communications network that could make use of any existing physical network. The second was that this combined network could also continue operating if one or more of the network nodes ceased to operate, as in the case of it being destroyed in an attack. The initial test network was known as the Advanced Research Project Agency's Network (ARPANET) and included test networks for radio and satellite communication.

As ARPANET grew, it was split into two networks. One was a dedicated military network and the other continued as ARPANET. The most significant step in the Internet's history was when DARPA decided to allow the distribution of their code for the IP suite of protocols. This allowed many universities and other institutions to join the relatively small test network. From that point, the original ARPANET expanded into the Internet as we know it today and the current version of the IP suite is version 4 with version 6 [Bra95] a topic of much debate.

C.2.2.2 How IP Works

The Internet Protocol considers every packet[14] of data being transferred through the network to be independent. At each network node along the data's route, the node makes a decision governing where the packet will go to next. The nodes make this routing decision for every packet, whether the previous packet had the same destination or not thus each packet is considered individually and packets may take different routes between source and destination. For this reason, IP is known as a connectionless-orientated protocol because there is no concept of an end-to-end connection.

Since each node makes the routing decision on a per packet basis, there is no guarantee that every packet will travel the same route to the destination. There is also no guarantee on the order in which the packets will arrive at the destination. The Internet Protocol itself is not concerned with these effects and does only its best to ensure that the packets get from one node to the next. It also defines the information needed to be stored in the packet in order for the node to route the packet correctly. This information is known as the packet header and contains information such as the source and destination address.

In order for the network node to make its routing decisions, it needs to have some way of determining where next to send each packet, depending on its destination. This information is stored in a routing

[14] A packet is quantity of data of set size that is sent as a single complete unit.

table. Initially administrators needed to update these tables manually, reducing the flexibility of network growth, but ensuring their correctness and not producing loops. If two or more routers were to cause a packet to travel in a circle, or loop, then the packet would never reach its destination and the routers would be wasting network bandwidth.

As the ARPANET project matured, the size of these tables began to get very large to accommodate all the possible routes they needed to know about for each destination. As these tables grew larger, they became more complex and more difficult to maintain manually. To solve this problem, special protocols were designed to allow routers to share information, configure and maintain themselves without human intervention, allowing the network to grow rapidly. This also permitted the network to respond automatically, to a degree, to a node ceasing to function, by routing around it using an alternate path. This increased the chance of there being a useable route to the destination, therefore increasing the apparent reliability of the network. However, the routers only offer a best effort delivery of each packet and this offers no guarantees about the time taken for the delivery of the data to the destination.

C.2.2.3 Transmission Control Protocol

Most communication on computer networks requires data to travel between a source and destination for a relatively long period of time and sending more than a single packet. If the data has to arrive at the destination in the same order as sent, such as a file transfer, then the transfer is done over what is called a *connection*. The applications using a connection transparently see a direct connection and if the source sends information, the destination receives it in order and without errors and the protocol supporting this type of connection is known as connection-oriented. The protocol on the Internet that provides this kind of service is the Transmission Control Protocol (TCP) [Pos81b], which operates over IP. It is one of the most common protocols used on the Internet and has been given the abbreviation TCP/IP.

TCP/IP provides a mechanism for simulating an end-to-end connection on top of an inherently connectionless network and provides services needed by a majority of applications while still maintaining the flexibility of the underlying IP network.

The main problem with TCP/IP is that it also inherits IP's "best-effort" design. As the amount of data traffic increases on the network, the frequency of congestion rises. When a network node is congested, it will drop packets that it is not able to process. If this packet happens to belong to a TCP/IP

connection, then it will have to be retransmitted by the TCP layer as it ensures that all the data will arrives at the destination. The retransmitting only makes the already congested situation worse.

C.2.2.4 User Datagram Protocol

TCP provides a way to support an inherently connectionless connection over IP. The User Datagram Protocol (UDP) [Pos80] is a simple protocol meant for providing datagram services to applications that do not require end-to-end connections. Since IP itself is already a connectionless networking protocol, UDP does not need to add much functionality. In fact, the only thing UDP does add is the ability to address virtual ports on the destination computer in the same way that TCP does. Other than that, because UDP is intended to act as a lightweight extension for IP, it does not offer any guarantees. It has no mechanism for checking that packets reach their destination and resending them if they do not, as in TCP, and therefore cannot guarantee packet delivery. It also cannot guarantee packet order. Since UDP uses IP and IP cannot give any guarantees about bandwidth availability, neither can UDP, they are both "best-effort" services.

However, one of the advantages of UDP is that because it is such a simple protocol it adds very little in terms of data overhead thus reducing the amount of time spent by the CPU on processing network packets. This time is therefore available for other tasks, allowing them to process data faster. If an application is more reliant on timely delivery rather than guaranteed delivery or ordering, then UDP is the better choice.

C.2.2.5 IP Broadcasting

IP Broadcasting [Mog84] is a simple way provided by IP for a single computer on a LAN to communicate with more than one other computer on that same LAN, without having to send multiple copies of the same packet. The protocol specifies an address, called a broadcast address that refers to all the computers within a particular logical network. For instance, to send to all the computers in the 10.128.24.* network, one would send to the address 10.128.24.255. Every machine with an address starting with 10.128.24 would pick up this single packet. This is a very simple way of sharing data with multiple computers. Its disadvantage is that every machine within the specified broadcast range has to process the packet, whether it is destined for an application on that computer or not. It is also limited to the Local Area Network.

C.2.2.6 IP Multicasting

IP Multicasting [Dee89] is a mechanism that allows an application to send a single packet to a group of end computers anywhere in the IP network. A single network address is assigned to identify a group of

end computers and when an application sends to that group address, copies of the packet are sent to all machines on the local network that are part of the group, and one packet is sent to a special multicast router. This router examines the destination address and determines if there are any other networks, or multicast routers, adjacent to it that have other end computers in the group. If there are, it sends a copy to each of those end machines and multicast routers.

In a switched network, this method is relatively efficient in its use of bandwidth. The only duplicated packets are for the end station on the same local network. Each multicast router then passes a single copy to each of the next multicast routers. This is shown in Figure C.2 and is often referred to as a multicast tree.

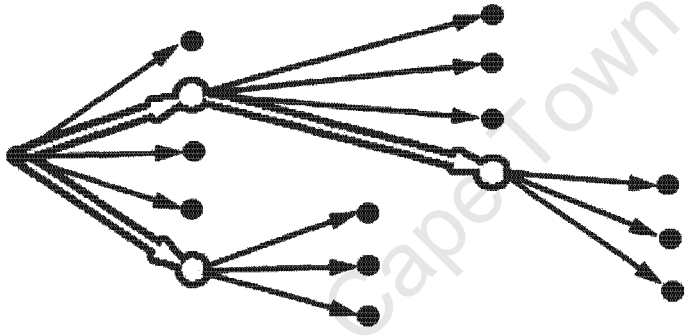


Figure C.2: IP Multicasting

The black dots, in Figure C.2 above, represent networked end-stations, while the highlighted dots represent multicast routers. The black arrows are the packets sent to the end computers on the local network, while the highlighted arrows show the single packets sent to the multicast routers. With the multicast routers in place at the source, only five copies where sent. The subsequent multicast routers were responsible for further packet distribution. While there are fewer packets sent, it is not the optimal situation which would only be a single packet being sent. It can easily be seen that the multicast routers improve efficiency by reducing the packet duplication needed because if those routers did not exist, then the source would have needed to send copies of the same packet to all the end stations (12 in this example).

The disadvantages with IP multicasting are similar to those of IP. There are no guarantees with regard to the information being sent. As with standard routers, multicast routers consider every packet individually, forcing multicast routers to examine every packet and to do a lookup on the destination to determine what action must be taken. This reduces the throughput capabilities of the routers. Further, IP multicast is not readily available as, due to legacy hardware, not many routes are multicast enabled.

C.2.2.7 Other Characteristics

The address space of the Internet Protocol is quite small, relative to other network protocols. This is one of IP's greatest problems. There is constant worry that the network is going to run out of address space, preventing the addition of new end-stations and nodes to the network [Bra95]. IP uses 32 bits, or 4 bytes, to store addresses. The number of combinations possible with 32 bits is in excess of 4 billion, but because of the way the addresses are broken up into classes and assigned in blocks, as shown in Figure C.3, it is very inefficiently used.

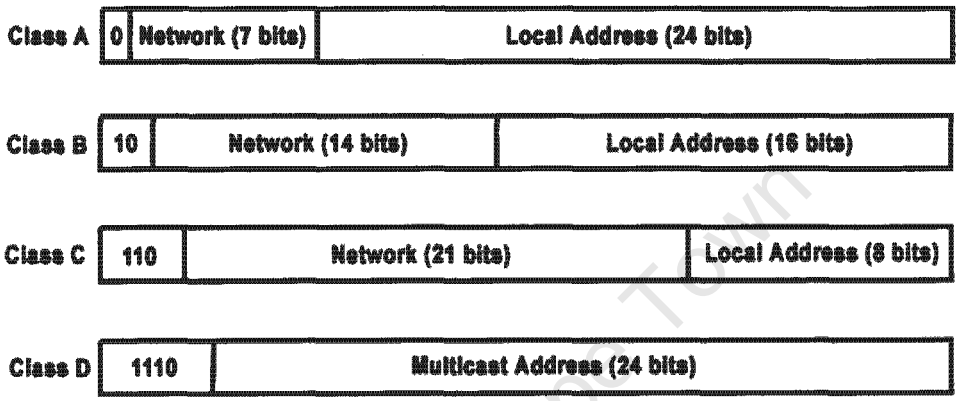


Figure C.3: The Four IP Address Class Structures

One of the primary concerns when groups worked on proposals for the next Internet Protocol was to extend the address space of IP so that, even with inefficient assignment, there would be more than enough addresses for many years to come. This next Internet Protocol was initially nicknamed "IP Next Generation" before being officially labelled IP version 6, or IPv6 [Bra95] [Qui00], while the older IP became known as IP or IPv4. This new version also tackled problems such as the lack of Quality of Service in the current IPv4, as well as extending the functionality of other protocols in the IP suite. It also added authentication and encryption information into the header options, enabling secure communication.

Unfortunately, in order to experience more fully the benefits of the improvements in IPv6, every network node along a packet's path needs to support IPv6. Since IPv6 is still not widely used, it is not possible to enjoy the improved functionality, such as QoS support. Another drawback of IPv6 is that, as with IPv4, it is still only a logical network protocol, requiring a separate physical network to transfer the packets. If the physical network cannot provide support for the services IPv6 offer, then IPv6 cannot guaranteed those services will work. For this reason, IPv6 is heavily dependant on the quality of the underlying physical network.

As an example, consider IPv6 operating over an Ethernet network. The IPv6 part could know the possible local maximum transfer rate of the connection, but since Ethernet offers no QoS guarantees, IPv6 can only try to simulate the support for QoS. If some other computer on the network were to initiate a large network transfer then the amount of bandwidth available on the network would drop without the IPv6 client having any control over it.

C.2.3 Asynchronous Transfer Mode

Asynchronous Transfer Mode (ATM) is a modern digital network technology capable of achieving very high bandwidths and very low latencies. Its designed uses include voice, video and general data, but by creating a high-performance base, it has enough flexibility to support any future applications as well.

About 80 percent of the service provider and telecommunications networks around the world use ATM in their backbone [ATM01]. This part of the network carries the largest amount of traffic with the requirements of high bandwidth, low latencies and high reliability. This domination currently makes ATM the worlds most widely used technology for backbones of both pure data networks, as in the internet, as well as telecommunications networks, such as phone and land-based television companies.

ATM was originally designed as the basis for the next generation of Integrated Services Digital Network (ISDN), namely Broadband-ISDN (B-ISDN). It aimed to allow full advantage to be taken of the potentially large capacity of fibre-optic media. One of the basic design ideas of ATM is that the majority of data transfers are over end-to-end connections and with this in mind, ATM was designed to very quickly and efficiently switch small, fixed-length packets, called cells, through the network along a previously specified path. This core simplicity allowed the network to scale very well, in both size and transfer speeds.

In addition, the design included ways for users to indicate their connection requirements to the network, in terms of bandwidth, latency, etc. If these resources were free, the ATM network would assign them to the user's connection, when it was being set up, and the user would have that Quality of Service (QoS) for the entire duration of the connection. This was vital to telecommunications companies that could not afford to have telephone calls cut off due to network over-use, and television networks that could not afford to have television pictures breaking up. In both cases, the network has to guarantee that nothing, especially network load, disrupts these services.

C.2.3.1 ATM Standards

The protocol defines a set of layers where each layer represents a different logical section of the full protocol standard. Figure C.4 is a diagram from the Broadband Integrated Services Digital Network (B-ISDN) model illustrating the ATM protocol layers.

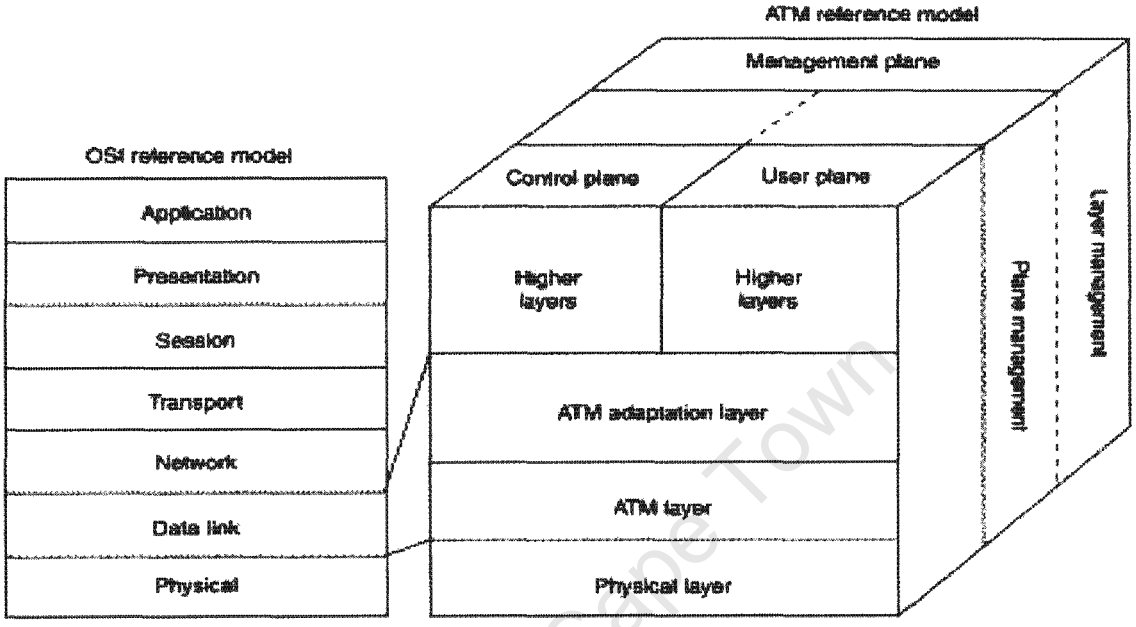


Figure C.4: The B-ISDN ATM Reference Model [Cis99]

- *Physical Layer*: At the bottom of the stack is the physical layer, which is further broken into two segments. The lowest governs the timing of the data sent and the way in which it is sent. This segment is heavily dependant on the medium used to communicate, whether it is copper wires or fibre-optic lines. The second segment accepts individual cells from the ATM layer and prepares them for sending, including generating the header error check CRC. It also processes received cells, stripping the extra framing information, checking the header against the CRC and then passing complete correct cells back up to the ATM layer.

- *ATM Layer*: this layer is the most complex and defines many things. Firstly, it defines the structure of the cells used to transfer data in the ATM network.

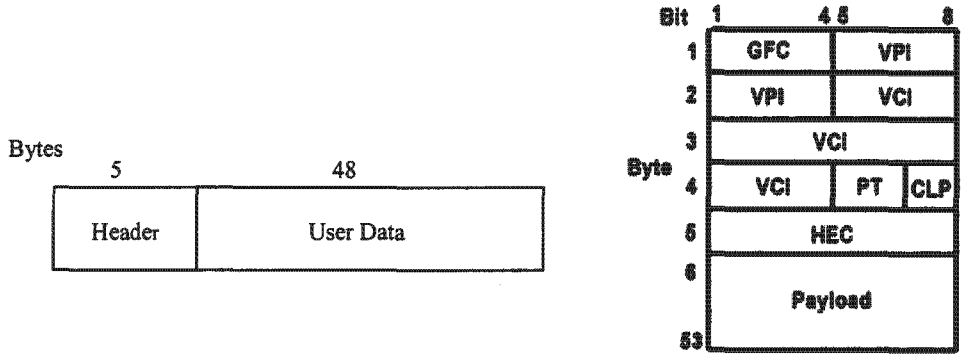


Figure C.5: (a) ATM Cell Structure (b) ATM User-Network-Interface Header Format

Every ATM cell is a fixed 53 bytes long and split into its header, 5 bytes, and payload, 48 bytes, as shown in Figure C.5a. The User-Network-Interface (UNI) format is the structure of the cell header used by end computers to communicate with the switch they are connected to and is shown in Figure C.5b. Once a connection has been set up, two unique identifiers are assigned to the connection, a Virtual Path Identifier (VPI) and Virtual Circuit Identifier (VCI). This VPI/VCI combination is placed in every cell sent into the network along with the Generic Flow Control (GFC) identifier used to provide a framework for flow control and fairness and is used to implement access levels and priorities. The Payload Type (PT) field indicates the type of data in the payload; whether it is user data or network management data that switches have to act on, or something else. The Cell Loss Priority (CLP) field gives an indication of the cells willingness to be dropped by the network. If the value is zero, then a switch is required to send it through. If the value is one, then the switch can drop it if it needs to, in order to save bandwidth as in a congestion situation. The Header Error Check (HEC) field is where the CRC, mentioned in the previous point, is stored.

This layer also defines how switches pass data through the network. When a cell enters a switch, the VPI/VCI combination is looked up and a new VPI/VCI combination placed in the header. Based on this new VPI/VCI combination, the switch will then send the cell on to the next correct switch in the connection path. This lookup and forwarding is done in hardware at a very low level, unlike IP routing, which still sits on top of whatever physical network is being used and is often done in software. Doing this in low-level hardware, close to the physical medium, makes the ATM switching process very fast, allowing ATM to achieve high data throughputs.

For the higher level application interface, this layer accepts larger blocks of data from the higher layers and breaks them up into cell payload sizes, creates the header needed, based on which connection it is for, and passes them onto the physical layer for transmission. Likewise, when it receives cells from the physical layer, it removes the ATM header and groups the various cells for a connection back together, passing the complete block of data up to the awaiting application.

- *ATM Adaptation Layer*: This layer adds extra functionality over and above that provided by the ATM layer. It is designed to offer specific types of service intended to support certain classes of application, such as video on demand, data transfers and voice, each of which have different network requirements and extra support not offered by the ATM layer.

For example, the ATM Adaptation Layer (AAL) for data transfers, number 5 (AAL5), includes extra information to ensure that data is delivered correctly. The header error check CRC is only a check for the ATM cell header to ensure that the cell is passed through the network correctly. There is no guarantee that the payload is corrupted. For this reason, especially for data transfers, AAL5 allows large blocks of data to be sent, breaking them up and adding, amongst other things, a full 32-bit CRC to check that the block of data arrives at the destination correctly. If it does not, it requests a retransmission from the source before sending the data up to the application.

Table C.4 contains a quick summary of the various standard AALs, including their number, short description and possible uses

.

AAL	Description and Application Type
0	This layer is considered *Raw* or *Native*. The layer does not supply the payload with any extra correction or overheads allowing for a full 48 bytes of information to be sent. The service categories can still be used with this category of service.
1	Used for real-time applications that require a constant bit-rate, connection orientated end-to-end data link. Four bits of the payload are used to store cell sequence numbers. This layer is suitable for uncompressed real-time voice.

AAL	Description and Application Type
2	Designed for variable bit rate traffic such as real-time delivery of compressed videos.
¾	Intended for connectionless data services where there is no timing requirement. Four extra bytes are used for control information
5	Specifically design for packet applications, such as Internet applications. It has better error detection and lower overhead than the other standard services.

Table C.4: ATM AAL Categories

For those applications that wish to use the ATM layer directly, bypassing any of the listed AALs, there is a user definable AAL number, namely 0. AAL0 is for applications that want to implement their own application specific AAL, or for applications which have no need of any AAL and the extra services they provide. This can have a slight performance increase if AAL0 is appropriate for the application.

C.2.3.2ATM Service Categories

The ATM layer also defines a number of Service Categories. These indicate to the network what the application is requesting for this connection. Often the various AALs are mapped to specific Service Categories since certain categories are well suited to certain standard AALs.

Service Category	Description
Constant Bit Rate (CBR)	This is the first category and has the highest priority in the ATM network. It is for applications that produce data at a very constant, and therefore predictable, rate or for end-systems that can justify reserving a full CBR channel for guaranteeing response times due to particularly strict end-to-end latency requirements. This category is often associated with AAL1 for transporting interactive audio (digitised voice or telephony), audio and video distribution (TV, pay-per-view, distance learning) and audio/video retrieval (video-on-demand, audio libraries).
Variable Bit Rate (VBR)	This category is one for which end-systems can benefit from statistical multiplexing and can tolerate a small random data loss ratio. This category is further broken down into two sub-categories, each of which has slightly different requirements. The first is real-time VBR (rt-VBR) which has the

Service Category	Description
	second highest priority in the network. This category is for Variable Bit Rate traffic; traffic which is varying, but reasonably predictable, usually because the application has information about the way in which it is producing the data. It also has strict end-to-end delay requirements. The second sub-category is non-real-time VBR (nrt-VBR). This is also for variable bit rate traffic, but does not have the same end-to-end timing requirements as rt-VBR, but does ensure that the variation in end-to-end delay for each cell does not change by much. rt-VBR is often used by native ATM voice transfer where compression and suppression techniques are used. It is also particularly useful for interactive multimedia applications. nrt-VBR can be used for data transfer such as time critical transactions required in airline reservations, banking, remote sensing.
Guaranteed Frame Rate (GFR) [Bon01]	ATM normally measures the transfer rate through a switch in terms of ATM cells. Applications using packet type protocols, such as IP, may require entire packets to be transferred at a set rate, rather than spreading the cells out over time. GFR is a recent addition to the ATM specification that attempts to compensate for this by supporting a mechanism for guaranteeing the connection's frame, or packet, rate. An application will most likely use this service in conjunction with the packet orientation of AAL5.
Available Bit Rate (ABR)	Any non-time critical application that is able to vary its emission rates, this category allows for a minimum cell based constant transfer rate, with limited timing requirements. After an ATM switch has allocated the bandwidth required for the higher priorities as well as this category's minimum cell rates there is a certain amount of bandwidth unused. The user can then have information regarding the amount of unused bandwidth sent back to them, allowing them to increase their sending rate should there be capacity available. There are usually mechanisms to ensure that the available bandwidth is shared equally among the ABR connections through the switch, making sure no single connection tries to take control of all the bandwidth that is left. This category is currently the only category in which the network gives information back to the user about what resources are available in the network.

Service Category	Description
	This service would often be used for LAN interconnection and internetworking and LAN emulation.
Unspecified Bit Rate (UBR)	This category is the lowest of the ATM categories and uses the bandwidth left after the rest has been reserved for the higher priority categories and currently sending ABR connections. It is called unspecified because it is not possible for the user to specify any bandwidth rating, or for the network to predict any such rating. The user can attempt to send data, but the data will only pass into the network when there is bandwidth available. There are no guarantees at all in this category and it is the least reliable. However, many applications are able to tolerate delays and cell-loss and are able to use this economical category of service. Applications such as file/text/data transfer, messaging, telecommuting are all examples of services that could take advantage of UBR.

Table C.5: Descriptions of the AAL Categories [Lam02]

Characteristic	CBR	Real-Time VBR	Non Real-Time VBR	GFR	ABR	UBR
Guaranteed bandwidth	Yes	Yes	Yes	Yes	Yes, Minimum	No
Real-Time transmission	Yes	Yes	No	No	No	No
Variable bit rates	No	Yes	Yes	No	Yes	Yes
Feedback from network	No	No	No	No	Yes	No

Table C.6: Summary of the Characteristics of ATM Service Categories

In a business environment, CBR and rt-VBR would most likely have the highest cost of use. This is because of their combined strict bandwidth and timing requirements and guarantees. Similarly, the lowest charge categories would offer UBR since there it makes no guarantees about the bandwidth available or whether the data will even reach the destination. The other ATM service categories would fall into the middle-range charge categories, with extra services increasing cost.

C.2.3.3 Quality of Service Support

On most networks, there is no way for a user or application to request specific connection characteristics from the network. This means that there is no way for the users or applications to control the characteristics of the connection. In these cases, the network does the best it can to deliver the data as quickly as possible. However, since there is no interaction with the network, there is also no way to restrict the number of users that are connected to the network at the same time, all possibly trying to send data at the same time as well.

As the number of users increase on such networks, the effective bandwidth available to each of them is less than the total divided by the number of users. This is simply because of the many resends that will be having to take place because of dropped packets due to network overload. At no time does the network prevent a user from trying to connect or send data, unless there is a physical problem with the switch or router to which the user is connected.

The ATM network protocol allows applications to request very specific services from the network, from service categories, as mentioned in the previous section, to characteristics such as bandwidth and latency, as well as reliability. If the network has the resources available at that time to support the requested characteristics, then the connection is set up. If the network does not have those resources available then it will not allow the connection. In this case, the application can either abort what it is doing or try to reconnect using a set of characteristics with a lower rating. Either way, once a connection is set up, the network reserves the required resources for that connection for its entire duration. New connections being set up do not interfere with already existing connections.

One might well ask what would happen if an application requested a low bandwidth connection and then tried to send at a higher rate. The ATM network protocol handles this situation. When a user requests a connection, the application passes the set of connection characteristics to the network in the form of a *Traffic Contract*. If the network accepts the connection, it then uses the *Traffic Contract* to monitor the behaviour of the connection. This is known as *Traffic Policing*.

If the switch receives data on a connection that does not fit into the characteristics requested for that connection, like trying to send at a rate higher than the reserved bandwidth, then it can take one of two actions. First, if there is a large amount of bandwidth still available, the switch will usually mark the data as being non-conformant. This means that further into the network, if there is a node that is experiencing congestion, it can drop that non-conforming data before any other data. The second option is that the switch could just drop the data without even considering whether there is still

bandwidth available that it might use. This second option is quite severe, but it ensures that people do not try to take advantage of the network and abuse it.

Since the application also knows the contents of the *Traffic Contract*, having defined it itself, it can minimise the chance of loosing data in the network by ensuring that the data it sends does indeed conform to the contract. This mechanism is often called *Traffic Shaping*.

The most common way of implementing traffic shaping is to place data for sending into a temporary storage structure, called a buffer, and then have a periodic timer that sends a portion of the data into the network, such that when the data is sent, it remain within the agreed upon traffic contract. While this mechanism results in predictable traffic, it has disadvantages. The extra step of copying the data to the buffer has two major effects. The first is that this extra step requires an additional memory copy, which although it is fast, does take time, and when many have to take place, the performance impact on a system can be severe. The second effect is that the data placed into the buffer may have to wait for the timer before the application will consider sending it. This waiting for the timer ensures that the application will not immediately send data placed into the buffer, incurring a further delay.

C.2.3.4 ATM Multicasting

In IP, there were two ways of sending data to multiple computers, broadcasting and multicasting. In ATM, it is also possible for one computer to send data to multiple others at the same time, but unlike IP, there is only one native method for doing this. ATM multicasting has similar base ideas as IP multicasting, where those that want to receive the data for the multicast group have to join the group.

ATM multicasting has a number of differences from IP multicasting however. In IP multicasting, the source machine itself sent multiple packets into the network, one for every machine on the local network and one for the multicast router. In ATM, this is not the case. The source machine only sends a single cell to the switch, which will then do any copying that is necessary, passing it on to local hosts or on to the next switches that require the cell. Since the switches only need to send at most one cell per link, the bandwidth efficiency of ATM multicasting is optimal.

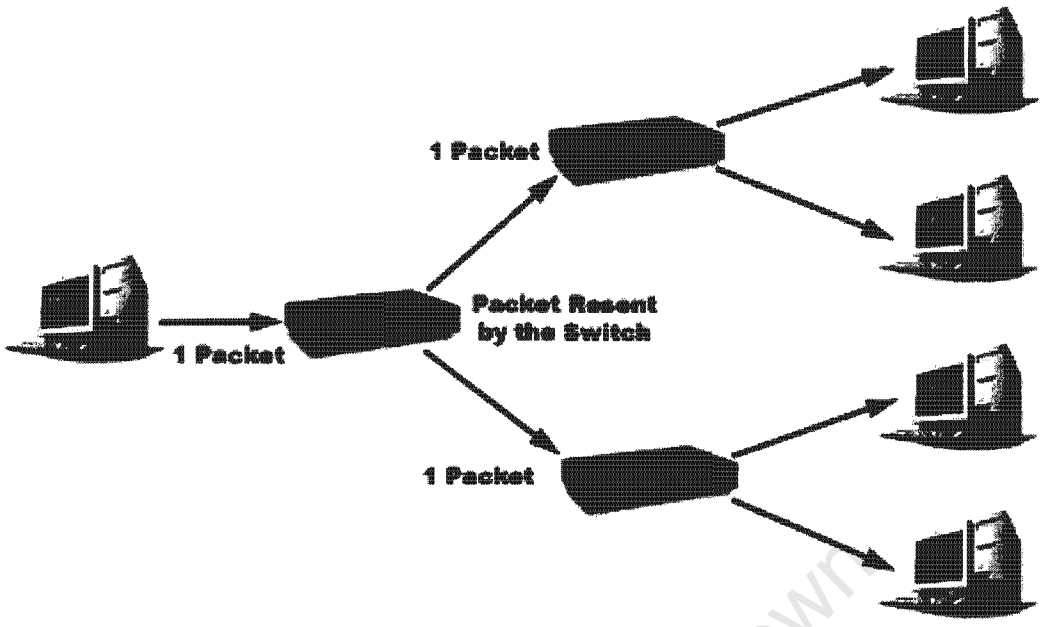


Figure C.6: ATM Multicasting

Another difference is that ATM multicasting supports QoS in the same way that single ATM end-to-end connections do. This can be a disadvantage in some cases, as the QoS needs to be the same for every user on the multicast group. If a user cannot get a connection to the multicast group with a QoS equivalent to QoS set by the owner of the group then they will not be able to join. This does however ensure that all members of the group will be able to handle the same data traffic, but in some cases where variable QoS is required, ATM multicasting may not be the most appropriate solution.

Until recently, ATM multicasting was limited to single direction connections allowing only the source of the "multicast tree" to send data to the other members of the group, otherwise known as the "leaves". This was the initial design of ATM multicast. Since then, the need became apparent to have multicast groups where everyone could send to everyone else. Initially this could just be emulated by manually creating separate multicast trees originating at every member of the group. This was inefficient since applications making use of the ATM multicast feature had to check multiple connections for data from a single group.

With this in mind, the ATM multicasting extensions were improved to include a mechanism to automatically create the multiple multicast trees within the network, and still have them appear as a single combined connection to the application. This improvement made ATM multicasting as useable as IP multicasting and far more bandwidth efficient.

A short example can easily demonstrate the effect of the bandwidth efficiency advantage of ATM multicasting. If a user dials into an Internet Service Provider (ISP) using a Digital Subscriber Line (DSL) connection then they usually have up to 640Kbits/s bandwidth for sending data. If the user joins an online game then the chances are good that a number of people in the game will be on the same network since ISPs usually have a relatively large client base and assign IP numbers within a set range, all of which are considered part of the ISPs network.

In this example, assume that there are 20 people in the game in the same area as the user is, and therefore the user joins a multicast group with them. Of these 20, 10 are on the same local network. If they are using IP then for the user to send one packet to the group at least 11 packets will need to be sent, 10 copies for the local users and at least one for a multicast router. This means that the user has effectively 640kbits/s/11 bandwidth to the group, i.e. 58.2kbits/s. However, if they were using ATM then the user would only have to send one cell. The first ATM switch would then duplicate it and send the copies on. This means that the user will have access to the full 640kbps sending bandwidth, significantly more than the 58.2kbps of the previous user.

Another restricting factor of ATM multicasting, often overlooked, is that ATM switches only support a limited number of ATM multicast connections through them, up to 4096 per port on modern backbone switches. This prevents there being a large number of long distance multicast trees, as could be found in widely distributed systems, such as television program streaming or large-scale multiplayer games. In order for a number of these applications to exist simultaneously, they will need to be designed to minimise all their multicast use.

C.2.4 Technology Comparison

Table C.7 summarises the major features of the various technologies discussed in Section C.2.1 with specific focus on the value that the services they supply can offer to a DVE design [Sin99].

Protocol	Strengths	Limitations	Net-VE characteristics
Ethernet	• Low overhead • Ease of use • Wide installation base • Simultaneous one to many possible	• Delivery only to computers on the same physical network • Extra overhead for computers not interested in broadcasts • No QoS guarantees	• Very useful for small scale local peer-to-peer Virtual Environments
TCP	• Guaranteed delivery • Ordered packet delivery • Packet checksum checking • Transmission flow control	• Point-to-point connections • Bandwidth overhead for larger packet header • Packets may be indefinitely delayed to preserve ordering • No QoS guarantees	• Virtual Environments having relatively small number of users and limited data requirements • Typically used in client-server systems
UDP	• Packet-based data transmission • Low overhead	• Point-to-point connections • No ordering guarantees • Packet corruption possible • No QoS guarantees	• Virtual Environments with higher data requirements • Used in both peer-to-peer and client-server systems

Protocol	Strengths	Limitations	Net-VE characteristics
IP Broadcasting	• Same as UDP, except supports simultaneous delivery to multiple destinations	• Simultaneous delivery limited to local logical network	• Small-scale peer-to-peer systems with high data requirements and time sensitive data delivery
IP Multicasting	• Same as IP multicasting, but delivery to anywhere on the Internet	• Similar to UDP Only available from/to computers connected to multicast enabled routers	• Large-scale peer-to-peer and client-server DVEs, particularly over the Internet
ATM	• Same as TCP • Less overhead than TCP • Guaranteed QoS	• Only supports point-to-point connections natively	• Similar to TCP but supporting a larger number of users
ATM Multicasting	• Same as ATM • Efficient delivery to multiple destinations on the ATM network	• Every user on the same multicast tree is required to make use of the same QoS • Limited number of multicast connections through switches	• Not clear at present, possible use in hybrid VE architectures

Table C.7: Summary of Current Network Technologies

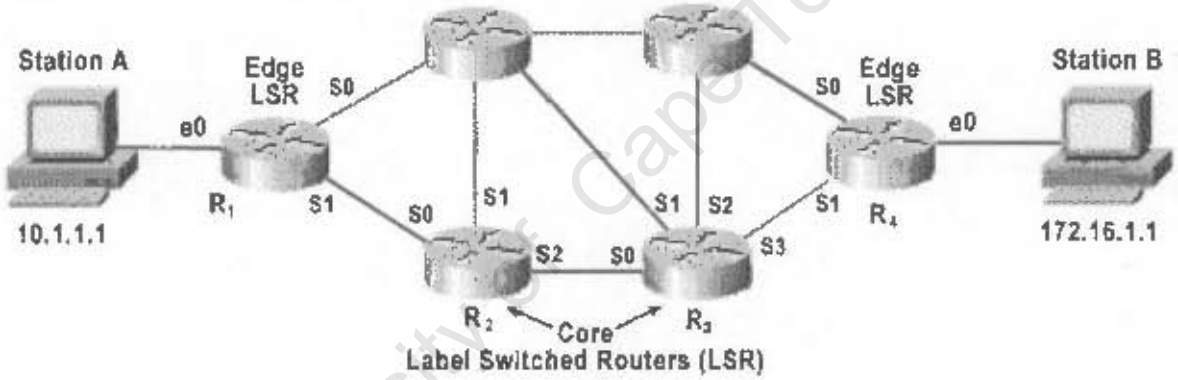
C.3 Convergence Technologies

The previous section focused on the major networking technologies currently available. Networks with mixed protocols have to spend time and processing power translating the various protocols and this is detrimental to the network's performance. Research is slowly allowing these technologies to converge into one protocol and to eliminate the overheads. This section aims to give a brief overview of the current and emerging convergence technologies.

C.3.1 Multi-Protocol Label Switching

Multi-Protocol Label Switching (MPLS) has the potential to be the most significant convergence technology yet [Ros01]. MPLS is a network layer enhancement that provides a connection-orientated service over a connectionless network and it is based on various vendor specific "tag-switching" enhancements to IP routers.

When an MPLS router receives a packet, it will look up the MPLS label in a table, using the label as an index into the table. The table will then tell the switch what to do with the current label, either modifying or removing the label, or adding a new one. The table will also indicate where to send the labelled packet next. In this manner, a packet will traverse an MPLS network, as shown below. In a way, this strategy is very similar to the ATM use of VPI/VCI combinations with table look-ups and as such integrates well with ATM. An example is shown in Figure C.7.



Router	Incoming Label	Incoming Interfaces	Destination Network	Outgoing Interfaces	Outgoing Label
R_1	—	e0	172.16.1	S1	6
R_2	6	S0	172.16.1	S2	11
R_3	11	S0	172.16.1	S3	7
R_4	7	S1	172.16.1	e0	—

Figure C.7: MPLS Path through the Network

The only major disadvantage with MPLS is that it does not have any support for multicasting. This is however being studied by the Internet Engineering Task Force (IETF), but for now it can make use of lower level mechanisms, such as Ethernet broadcasting, to achieve the same effect on a limited scale.

The advantages of MPLS for connectionless-orientated networks such as IP are many. Firstly, since a label is used as an index directly into the table, the look-up and replacement is fast in comparison to

the routing table lookups. This alone offers a great improvement in performance on connectionless networks. It also improves the manageability of connectionless networks. As one example, packets with differing QoS requirements can be assigned different labels upon entering the MPLS network, making bandwidth management easier, by not having to reinterpret QoS requirements for every packet.

MPLS provides a means of integrating connection and connectionless-orientated networks thus allowing it to create connections that can transparently span a heterogeneous network cloud, making efficient use of the native networking technology at each stage.

C.3.2 ATM over Ethernet

A relatively new technology that has been standardised by the ATM Forum, ATM over Ethernet [ATM00a] is intended to enable machines on Ethernet LANs to use ATM services, effectively emulating an ATM network on top of an Ethernet physical network. This can ease the migration from an Ethernet environment to native ATM as well as providing a way to integrate ATM and Ethernet LANs. Since ATM natively supports QoS, while Ethernet does not, this standard is based upon the principle that it is possible to guarantee QoS on an Ethernet LAN if there is enough total capacity relative to the total amount requested on the medium. If the applications behave reasonably well according to their agreed contracts, then the illusion of QoS can be maintained. However, if they do not, then the QoS of every user could be severely jeopardised.

C.3.3 Frame Based ATM over SONET

Another new technology, also from the ATM Forum, is Frame Based ATM over SONET (FAST) [ATM00b]. FAST is aimed at ATM packet services, such as AAL5, and provides a way to transmit variable length packets while reducing the cell header overhead, improving the efficiency of the link. It does this by removing the normal ATM headers and replacing them with another header only at the start, which gives the length of data being sent. This data is then spanned across multiple cells, all of which are sent down the link in series, typically with no interruption from other connections. For small data packets, the efficiency is reduced due to the extra header size. This standard can be seen as an extension for ATM that more efficiently supports packet services such as AAL5 and hence IP traffic.

C.3.4 Multi-Protocol Switches

As the major technologies become more entrenched and mixed environments become more necessary due to different service requirements, the need for more general switches increases. To meet this demand, various multi-protocol switches have been produced that can support a variety of physical and network layer protocols, usually through the use of appropriate pluggable network modules. These

hybrid switches are often used as high-speed gateways between multiple ATM and Ethernet networks, or as edge routers interconnecting a number of Gigabit Ethernets with high-speed ATM uplinks that join to an ATM backbone. There is no direct relationship between MPLS and multi-protocol switches, but such switches may provide MPLS support as it would allow it to handle data packets in a low-level, general manner. This would also improve a switch's efficiency when forwarding data between ports.

C.3.5 Digital Subscriber Line

The last new technology introduced is Digital Subscriber Line (DSL) [Cis99]. This is an access technology, typically for home subscribers, but also used by many businesses. DSL makes efficient use of copper cabling, from the site of the telephone to the local telephone exchange, allowing higher bandwidth network access. Currently, the majority of the deployed technology is Asymmetrical DSL (ADSL) [DSL01], and is to homes. ADSL has two communication channels, on one it supports between 1.5 and 6Mbits/s downstream[15] only, while on the other it supports up to 640kbits/s in both the upstream[16] and downstream directions.

Since DSL is a physical transport technology, it can run any networking protocol over it: Ethernet, IP or ATM. ATM over ADSL was the first transport mode standardised for ADSL, with Packet over ADSL coming later, providing a means to support IP directly over the ADSL link. Ethernet is also beginning to make headway into the DSL arena with a proposal for Ethernet over Very-High-Data-Rate DSL (VDSL) [Bar01].

C.3.6 Summary

As can be seen, there are a number of emerging technologies that are bringing together different aspects of the various major network technologies and extending them all the way to the home-user. These new technologies offer increased interoperability and ease of migration from one standard to another, while making way for a single next generation networking standard as the major technologies converge on the same goals.

[15] Downstream is from a remote source to the user.

[16] Upstream is from the user to a remote destination.

Appendix D: Virtual Reality Equipment

Appendix D contains a brief overview of the various tools that one can use to navigate, simulate and experience the reality of VE. While some might consider this information unnecessary, it is important to know equipment is available and the quantity of information a tool needs to function correctly or generates when in use. This knowledge needs to be incorporated into the design of the VE or the VE might not be able to support the tools and not offer a believable experience to the user.

When considering devices, it is important to consider all five senses. The senses of sight and sound are often the only senses considered when referring to output devices associated with computers, but technology is now available to simulate smells, the texture of objects that can be touched, and while the sense of taste is not easily simulated, there is work in progress to simulate this sense too. Input devices are also limited, but tools have been created to track the body's movements, the focal point of the user's eyes and the speed and force with which a user moves.

D.1 Graphics

The aim of a graphical device is two-fold. The graphical *engine* generates images from the data about the environment, and the display produces the visual interface for the user. The capabilities necessary to model and display life-like environments have in the past, only been available on high-end graphical workstations such as those made by Sun [Sun02] and SGI [Sil02]. This trend towards the more powerful systems has slowed due to the rapid development of graphics hardware that gives personal computers the ability to display detailed environments. Developments in graphical standards, such as OpenGL [Sil02], have also allowed cross-platform software development. This has sped up the development of graphical hardware and allowed more users to become accustomed to the graphical interfaces available and ensures a high demand for the developed hardware.

D.1.1 Displays

The display, in its various forms, is a key tool in creating a believable environment. The display is used to impart a large portion of the information about the user's environment. While a monitor can provide high-quality images to the user, it only offers a limited sense of realism as the user can be easily distracted by peripheral occurrences. The quality of these images is also important and the

higher the resolution[17], frame rate[18] and colour depth[19], the more realistic the image can appear, and the more immersive the environment will seem.

To help increase the realism, 'shutter glasses' can be used. Shutter glasses block off the user's eyes alternately and is synchronised with the monitor displaying two slightly different views of the environment and this creates the illusion of a three dimensional picture. To support shutter glasses, the graphical hardware needs to be able to produce video frames at twice the rate that it would normally need to. While the shutter glasses do offer increased 'realism,' the user is forced to remain looking at the monitor and the situation still suffers from distractions.

Figure D.1: (a) The nVision Datavisor HMD (b) Olympus Eye-Trek FMD-700 [Nor02]

---

17 A pixel is the smallest dot that can be drawn on the screen and an "x" and "y" coordinate and a colour value represent it. The display's resolution is the maximum number of pixels that can be displayed at one time and is usually given by listing the maximum number of rows and columns. The higher the resolution, the more detailed the picture can be, but as the resolution increases, so does the load on the system. High-end displays often support 1280x1024 while computer-generated images for the entertainment industry are often created at 4096x4096.

18 The *frame rate* is a measure of how rapidly the system can completely update the entire screen and is usually measured in Frames Per Second (fps) This is an important metric as the frame rate is a measure of how smooth the motion in the pictures appears to be. The converse is that the more frames you want to display, the faster the system will need to render and display the frames, putting a larger load on the system, especially for high resolution, high colour images. The standard for most modern computer games is 30 fps as is the television, while the cinema runs at 24 fps.

19 Colour depth is a measure of the number of colours that a pixel can take on. The current standard is labelled '*True Colour*' and consists of 32 bits of information with 8 bits being used for transparency. The colour depth could be further increased but this is unnecessary as the human visual system has trouble distinguishing colours that are too close to each other and an increase in the colour depth will just add unnecessary load on the system.

Head-Mounted Displays (HMDs), examples of which can be seen in Figure D.1, consist of small graphical displays positioned before the user's eyes in a goggle-like visor. The display show the user two slightly differing views of the environment and creates the illusion of a three dimensional picture. Further, the user sees a much wider field of view[20] while blocking out almost all external light and other distractions. The user is also then able to move their head without interrupting the realism of the environment. Often the HMD has position sensors and headphones, the former an input device giving information about the user's head position and the latter an output device for sound. Both will be looked at in later sections.

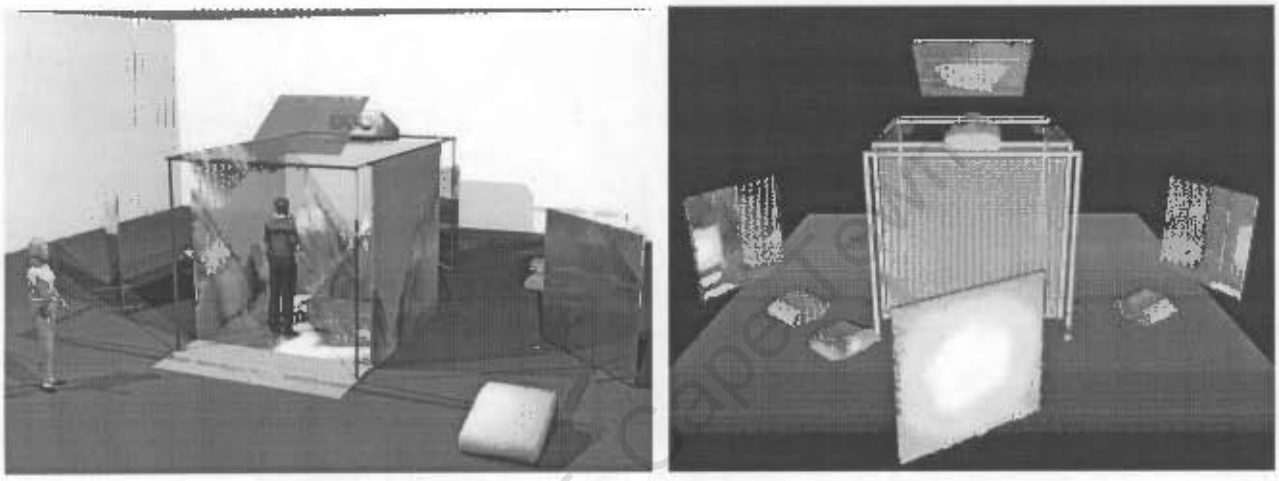


Figure D.2: Idealised Views of CAVE Automatic Virtual Environments

Another technique, examples of which are shown in Figure D.2, is another option for displaying an environment. Users stand inside a projection cube where projectors project images onto the walls, floor and ceiling of the 'CAVE Automatic Virtual Environment' (CAVE) [EVL02] [Fak02]. The images are synchronised to shutter glasses worn by the user, creating the illusion of a three dimensional environment in which the user can move around and in which the user has a large field of view.

20 The *field of view* determines how much peripheral vision the user has. Looking through a telescope gives you a very narrow field of vision, while looking at a 'widescreen' TV gives you a large field of vision. The larger the users' field of vision, the stronger the feeling of 'immersion' within the environment as the user feels that they are surrounded by the environment instead of peering into it.

D.1.2 Models and Images

The environment is constructed from many 3D models of objects and the detail displayed in each of these objects affects the realism of the environment. Each model consists of many polygons that are joined together to form the object. Since most objects have curved surfaces and the polygons used to create these surfaces are 2D planes, the curved surfaces have to be approximated using many small polygons. The more polygons used, the smoother the object's curved surfaces will look. This can have an inverse effect on the frame rate, and thus the realism, as the greater the complexity of the scene viewed, the larger the number of polygons needed to create the scene, and the longer it takes to render the scene, thus significantly reducing the frame rate.

The realism of the scene viewed can also be increased using several other techniques such as texture mapping, where 2D pictures are pasted onto an individual polygon within the scene. The polygons are then not just one colour but textured to look like the picture. This gives the impression of more detail than is actually present in the 3D model. Lighting is another technique that is used to shades the polygon to simulate light sources, and transparency that allows the user to see through the surface of an object and simulates such things as windows and other fully or partially transparent object.

D.1.3 Adapters

The graphical adapter is clearly an important factor in the creation of the images that are displayed in a VE. Modern graphical adapters are able to reduce the time taken to render a scene by accepting the raw data from the processor and using dedicated hardware to do the processing. Before this was possible, the rendering of the scene was done in software and computed on the CPU. This took time away from other important processes.

The 'acceleration' process accepts the coordinates of the polygons, their textures, any light sources, and the user's viewpoint[21] within the scene and renders them on dedicated graphics processors using dedicated memory, sitting on the card. The calculations on the hardware, being dedicated to graphical processing, are much faster than if the CPU itself performed these calculations.

[21] The viewpoint is the location from which the viewer is looking at the scene. In a 3D VE, the viewpoint matrix is applied to all the objects in the scene to determine where the objects are relative to the user's position and orientation.

D.2 Audio Devices

Hearing is a very important sense as we often locate and identify what is happening around us based on the sounds that are made. Sound is also simpler for computers to simulate than graphical content, as the quantity of data[22] is far less.

The 3D nature of a VE makes it possible for the audio to be altered from a simple 2D source so that it appears to come from different positions in the three dimensional environment and thus offering further realism to the user. The process of modifying the sound is called *convolution* and, much like the processing of graphical content, has been moved from the main system processor to the audio device's dedicated audio processors and memory. In addition, standards are currently being developed to allow users to use these abilities across various platforms.

For the user to feel immersed in the audio environment that is being created, the user often wears headphones that can also be incorporated into the shutter glasses or HMD. Another option is to use multiple speakers and surround the user and this technique is often used in CAVE situations.

Sound can also be used as an input device allowing users to communicate with others in the environment via a speaker system or using voice coded commands to activate functions within the VE.

D.3 Control and Input Devices

Users need to be able to manipulate and interact both with objects in the environment and the environment itself. They also need to be able to communicate verbally, texturally or graphically with each other as well as being able to move around within the environment. To do this, the user needs to use a device or devices that will make these types of interactions possible.

- Various input devices have been designed and the simplest and most common of these are the mouse and keyboard. While these two devices are common, and can be used for many purposes from communication to movement within the world and item interaction, they are not

[22] There are two important factors when dealing with sound: The first is the sample rate which is a measure of the number of samples taken per second and is measured in Sample per Second (sps). The sample rate is limited practically by the human auditory system to about 44100sps. The second is the number of bits used to encode the sound each time a sample is taken. This is measured in Bits per Sample (bps) and CD quality sound uses 16bps.

the most ideal as they are designed for a two dimensional environment and do not create a 'realistic' impression of the interaction with the world.

- "Joysticks," "microphones," "wheels" and "pedals" are devices often used for games and vehicle simulators where they are able to simulate the actual devices used in the 'real' situations that are being modelled in the VE.

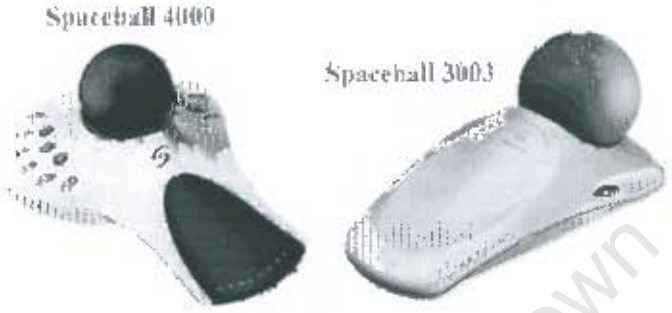


Figure D.3: 3D Connexion's "Spaceball" Series of Force-Balls.

The "force-balls" in Figure D.3 allow a user to manipulate a pointer in three dimensions while offering a number of buttons for completing various actions on objects once they have been selected.

- Body covering suits or gloves use sensors to monitor the motion of body parts. While the suits and gloves produce large quantities of data about the user, the data can be overwhelming. Using these tools, the user is able to use their body in a fully 3D way to interact with non-static models rather than with a simple 2D, 3D or other pointing device.

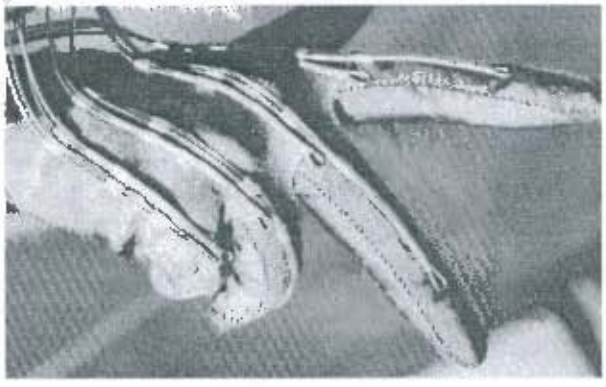


Figure D.4: The Greenleaf Medical System's CyberGlove

The "dataglove" in Figure D.4 is an example of such a body covering sensor array. The glove monitors the position of the user's hand relative to some fixed point as well as calculating the

user's wrist and finger movements. The results are translated into a model of a hand that can be viewed within the environment.

Figure D.5: Ascension Technology's MotionStar Sensors [Asc01]

- Magnet or ultrasonic trackers are often used to track necessary body parts as shown in Figure D.5 where an astronaut's body is being tracked during training. The trackers calculate their position relative to a fixed point. An example of where such a tracker could be used in, besides the dataglove shown above, is an HMD where the sensors are used to calculate the user's viewpoint.

- Visual and motion trackers are becoming more popular and work on the principle that objects within a frame can be identified and their motion calculated through consecutive frames. Motion sensors can be mounted around the user and while this is limiting, it is useful in an environments such as a CAVE. Another option for visual trackers is to track the focus of the user's eyes and the focal point is used to determine which objects in the environment are being focused on. Yet another option is for the visual sensory to track the gestures of the user, translating them into gestures that the user's avatar can perform. The gesture recognition can be particularly useful as a tool for the communication for those who are unable to speak.

- Tethering systems connect the user to a fixed monitoring system that is able to detect the user's motion by calculating the forces applied on the tethers. This is often very restrictive to the user's movements, as they cannot move beyond the limits of the tethering system, but this is particularly accurate should the accuracy be needed.

Full Body Immersion is a term used when the user's body is fully tracked and their movements displayed within the VE. This type of immersion is not common as the suit itself is highly expensive and the user will need a high-end system to process the data that the suit produces. Partial Immersion could take the form of a single dataglove tracking one of the user's hands, and a tracker in the user's HMD to track their viewpoint.

With the current technological advances, devices are becoming more accurate, more encompassing and less intrusive to the user, making the user's experience of the environment more believable and increasingly immersive.

D.4 Haptic and Tactile Feedback Devices

Modern interface devices are becoming increasingly more realistic in their interaction with the user. One method used to increase this realism is known as haptic feedback and is the simulation of the sense of touch. By using an interface device that can apply gentle forces on the user, the sense of touch is simulated. Textures, friction, vibrations and solidity of an object can all be simulated, to a greater or lesser extent, by applying a force on the user's body. While this might sound relatively simple, consider the forces that would need to be applied to the human body that is simply sitting on a simulated chair! These forces can be immense and precautions need to be taken to ensure that the devices are not able to harm the users.

Simple devices such as force-feedback joysticks, wheels and glove-exoskeletons are becoming more commonplace as the technology is being embraced by the entertainment industry. Simulator games use force-feedback wheels to simulate the forces on the driver as they turn corners, or joystick to simulate the forces on a pilot during flight. These devices use various techniques from simple rheostats to alter the resistance to moving a wheel for a driving simulator, to systems of pulleys and motors that simulate how solid a surface is or small speakers that simulate textures in haptic feedback gloves.

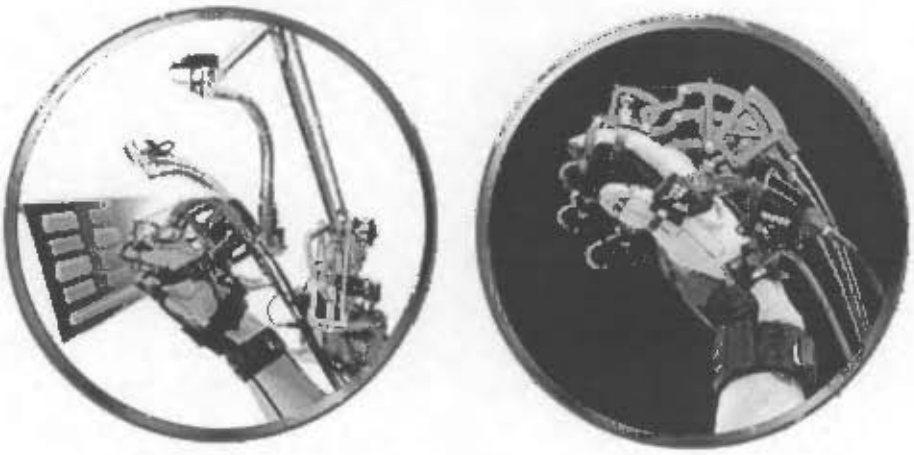


Figure D.6: (a) Immersion's CyberForce (b) Immersion's CyberGrasp [Imm02]

Figure D.6a shows the *CyberForce* haptic feedback device that can simulate the feeling of weight and inertia of objects or solidity of a virtual desk on which the user could easily rest their hand and arm. The system also allows full six degree of freedom[23] tracking of the hand and arm. The *CyberGrasp*, shown in Figure D.6b, allows force feedback for each finger in the hand and, combined with the *CyberForce*, can offer the user an extremely realistic feel to the environment.

Figure D.7: (a) Immersion's CyberTouch (b) Virtual View [Imm02]

CyberTouch shown in Figure D.7a uses small speakers that can be used to vibrate the user's fingertips and simulate the tactile texture of simulated objects as shown in Figure D.7b.

[23] The number of Degrees of Freedom (DOF) is a measure of the number of axes of motion in which an interface device can move. The first three degrees are considered the three-dimensional positioning of the object and the second three being the rotational motion, which includes pitch, roll and yaw.

D.5 Simulation Software

VEs require specific graphical engines that not only render the scene being viewed, but keep track of the models being displayed in the environment. This task is broken into two components known and managed by the Graphical Engine and the Scene Manager. The software available to do this is often rather expensive, as the research involved in the creation of these software packages is time consuming and costly.

When considering which graphical software package to use in the design of the DVE, both the cost and usefulness of the design must be considered. If the package is not able to deliver the support that the design requires, then the design will be significantly limited and may not be useful. Another point is that the commercial software does not often allow the designer to alter the code of the graphical software and this can limit the design when the graphical software does not supply the necessary services to the VE.

Name	Manufacturer	Approximate Cost	Release Date	Reference
WoldToolKit V8	Sense8 Corporation	$3,500	Late 1998	[Sen98]
Maverick	University of Manchester	Free (Open Source)	March 2001	[Mac02]
Code Creatures	Code Cult	Undisclosed	Mid 2001	[Cod02]
Genesis 3D	Eclipse Entertainment	$10,000 (Free for Research)	Nov 2000	[Ecl02]
Phoenix	4x Technologies	Free	June 2000	[4xT02]

Table D.1: Cost, Release Dates and Manufacturers of Graphical Engines

Table D.1 above shows a list of a few of the more well know graphical software packages that can be used in the design of a VE. The entry of note is the Maverick package which was recently released and is open source and allows a VE designer to alter the graphical software if need be although it was only recently released.

D.6 Hardware Data Rates

An important factor to note in designing a DVE is how much data the user needs to be able to view the environment realistically with the output devices and how much data is being produced by the input devices being used.

Device	Rate	Description
Generic Keyboard	480 bps	At a maximum of 30 keystrokes per second at 2 bytes per keystroke, the maximum data rate is 480 bits per second
Generic Mouse	2.400 kbps	
Immersion's CyberGlove	115.2 kbps	The glove has 18 sensors from which it records the hands and finger positions in three dimensions
CD Quality Audio	705.6 kbps	44100 sample per second at 16 bits per sample
Video: MPEG1 - NTSC	30.4 Mbps	352x240 resolution, 12 bits per pixel at 30 frames per second
Television: MPEG2 – NTSC	124.3 Mbps	720x576, 12 bits per pixel at 30 frames per second
Monitor	460,8 Mbps	800x600, 32 bits per pixel at 30 frames per second – A particularly large quantity of data
Head Mounted Display	460,8 Mbps	Same as the display output of a computer.
Trackers (3 DOF)	9.600 kbps	
Trackers (6 DOF)	9.600 kbps	This will also cover such devices as Forceballs. Updates occur +/- 60 times a second using RS232 at 115.2KBaud [Pol02]

Table D.2: Raw data rates of various input and output devices

Appendix E: Virtual Reality

'Virtual Reality' is seen by many as a 'modern' concept that has only found relative fame over the last few years with the release of hit novels and movies, such as *Neuromancer* [Gib85] and *The Lawnmover Man* [Leo92], but VR and VEs have been around for quite some time. In 1962 Ivan Sutherland created the *Sketchpad*, the first pointing device [NCS95a] for computers and in doing so, inspired the field of computer graphics. By 1968, Sutherland had created the first HMD, which allowed an immersive graphical interface for a computer.

The onset of World War II spurred the United States Military to spend large sums of money in the field of VEs and computer simulation to simulate aeroplanes and later tanks and ships. These simulators were needed to train pilots cheaply and safely on the ground before subjecting them to the hazards in the real equipment. Early simulators consisted of mock cockpits on motion platforms but were limited in their feedback to the trainees until the cockpit windows were successfully linked with video displays.

By 1970, the video visuals were replaced with computer-generated graphics that, while primitive, operated in RT. The military were by this time already experimenting with HMDs and simulators were becoming highly detailed with pilots able to fly through VEs that could test their abilities extensively.

The military was not the only industry interested in computer graphics. The entertainment industry was soon to appreciate these advances. By the mid 1970s, movies such as *Starwars* (1976) [Luc02] and *Terminator* (1984) [War02] were making extensive use of computer-generated special effects. Along with these special effects came innovations such as the dataglove. This device was a glove tooled to detect hand movements and convert these movements into data that could be used to simulate the more lifelike actions that were needed to simulate the various creatures and people. The entertainment industry also brought about the change that resulted in children all over the world spending hours in arcades playing computer games inspired by this technology.

Science has also gained extensively and is now able to use computers to visualise huge quantities of data and displaying them. Some of the many uses have been the mapping of the DNA sequences [HGP02], predicting weather patterns and mapping the notable stellar bodies in the universe around us.

The combination of scientific models and the entertainment animation led to the development of computer animation. Animation, while tremendously useful, still had huge limitations in the late 1980's and early 1990's because each frame in an animated sequence needed to be rendering and this could often take hours. This was costly both in terms of the price of the equipment and the time taken to produce the frames. The animations could also not be interacted with or altered and any changes would mean a completely new set of animated sequences to calculate. Even in the field of animation, interactivity is still as highly prized as in the fields of military and entertainment.

The final steps are current history where VR and VEs are now being incorporated into many areas of commerce, science and the military.

Area	**Uses**
Military	DVEs have been used extensively for both the testing and training of military personnel [Mac02] [Kar97] and for the simulation of various operational situations. These situations could include mission planning and simulation of the outcome should certain actions be taken and certain weapons be used. Of late, the military have even been using commercial available gaming systems as the basis for their tactical training needs [Dun02] and have customised commercial games *Operation Flashpoint* and *Steel Beasts* for their use.
Science	Scientists use the DVE platform for many varied tasks. Such tasks as weather simulation, atomic structure simulation and DNA decoding [HPG02] have all used 3D environments to enhance the scientists' ability to visualise the data and manipulate it in a more realistic manner. Even chemistry models are more effectively displayed in a VE than with the limited capabilities available in HTML and XML [Cas98].

Area	Uses
Design	Several companies, including *Caterpillar* [Cat02], *Boeing* [Boe02] and Volvo [Pyc00], are using collaborative three dimensional design tools to aid them in designing new aircraft and vehicles collaboratively and over large distances. Further, the designs can also be tested using the DVE by placing drivers or pilots in the virtual design and using the workers' experience to enhance the usability of the designs before construction. Architects have also used this technology to model designs for their customers who are then able to make changes to the designs before the actual construction is started. This saves time and money for both the architect and the customer.
Communication	Video conferencing has become a useful tool in today's business. Virtual conferencing takes this a step further by creating a 'layer' that allows those in the conference to use tools such as white boards, projectors and other board room apparatus within the virtual boardroom while still offering conferencing facilities such as the conventional 2D video feeds [Bil99]. Further, the voice input of the various users could be automatically translated before being sent to the other users. Similarly, a user who is speech impaired could use their own input device to translate hand signals into speech, which would automatically be 'spoken' to the other users and allow the impaired user to interact seamlessly with the other users.

Area	Uses
Teaching	Distance learning is another form of communication that has become highly popular, especially when the students and teacher are not able to meet due to various reasons. The teacher and pupils can meet in the DVE and the teacher, using virtual tools such as projectors, blackboard and models, can teach the pupils and reach a much larger number of pupils who might otherwise not have the chance of having the expertise of the teacher at hand [Mat97a] [Kim01] [Del02]. Medical training is also an important use for DVEs. Doctors, and doctors in training, are able to operate on virtual subjects to hone their skills. Further, with the aid of the modern scanning equipment, the VE can be created specific to each subject and a tricky operation could be executed beforehand on the virtual model and thus prepare the surgeons before they attempt the surgery.
Entertainment	VR has not only inspired stories such as William Gibson's *Neuromancer* [Gib85] but the technology has already been put to good use in the creation of realistically animated movies such as *The Lawnmower Man* [Leo92]. Scenes are also created to fit seamlessly into the more conventional movies we see today such as the ever-popular *Terminator 2* [War02]. This allows entertainers to create special effects and movie sequences that would never be able to be duplicated otherwise. Further, using multiple cameras and by combining the data, 3D video can be created of real life situations and viewed in a VE. This type of interactive television is on the forefront of research [Fla01]. The gaming industry also uses this technology for many popular gaming worlds: Asheron's Call [Ash02], EverQuest [Eve02], Quake [Qua02] and other similar games that allow gamers to interact with the environment and other users in the setting of their choice from fantasy to futuristic settings.

Area	Uses
Commerce	Online shopping complexes have become very popular as shops attempt to decrease the cost of offering their wares to the consumer. The virtual shopping complex allows the customer to select and purchase items online and have them delivered to their homes at a far cheaper rate. This is partially because of the reduced overheads that a warehouse offers in comparison to those incurred by a cosmetically appealing store.
Remote Control	The control of machinery in dangerous situations often still needs a human to direct the actions. These situations, such as mining and in nuclear power plants, can be very dangerous to the human body. Using a DVE to visualise the situation and then manipulate the machinery through the DVE removes this risk. An example of remote monitoring is the Virtual Environment for Network Monitory (VENoM) [Cub98] used to monitor and visualise the network status of an ATM network.
Medicine	Medicine is already making extensive use of VR and VEs in the training of doctors. *The Visible Human Project* [Nat01] has created a highly detailed 3D model of the human body and this model can be used to represent the human body realistically along with all the internal functionality for doctors and students to use when *practicing* and *examining* [Mor96] [Ber01a]. Other examples of VEs in medicine can be regularly found, such as the use of VEs to significantly reduce and control levels of pain in burn victims [Hof02].
Other	There are many other uses for VEs that do not necessarily fit into the above categories. For example, the REVEAL project is designed to recreate crime scenes from video and picture information [Gib00].

Table E.1: Areas where DVEs can and have been used to great effect

Appendix F: Summary of the HEAVEn Components (Design at a Glance)

The following is a summary of each of the components of the HEAVEn DVE as illustrated in Figure 4.10 which has been duplicated here for convenience as Figure F.1. The various components are described in terms of their functionality, interconnections and connection types.

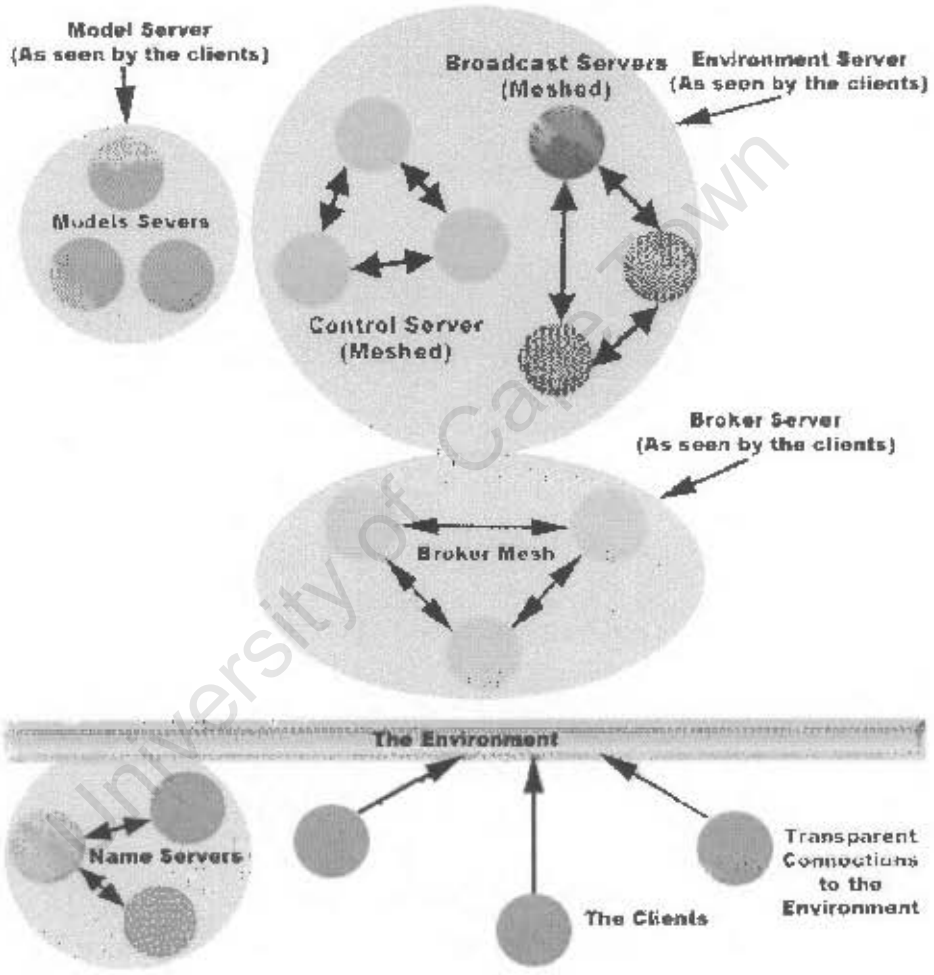


Figure F.1: The final HEAVEn architecture

Name Servers (Abbreviated: NS)

Main Functions:	A NS's main function is to translate *Descriptive Names* into *Lists* of ATM Addresses. The NS can be used to provide this service to other applications as they are not environment specific. The NS must be able to accept server registrations and de-registrations at servers boot-up and shutdown. Additional functionality can also be included to balance the load between the listed servers, to check the activity of the entries in the name database, and to log statistical information about the activity of the various named systems.
Server Interconnection:	NSs may connect to each other to share address information. This is often done using a hierarchical distribution of the servers to distribute according to geographical locality of the addresses.
Connection Types:	The connections are all AAL5 based ABR or UBR connections with a lightweight application level acknowledgement protocol to ensure data delivery.

Table F.1: Name Server Characteristics

Broker Servers (Abbreviated: Broker)

Main Functions:	The Brokers Servers provide a 'lookup' service that ensures a transparent interface to the DVE architecture by storing the addresses of the various servers in the environment. In addition, the broker store lists of available models and the MSs where these models can be retrieved. Both the servers and clients can connect to the brokers to retrieve address lists for the various servers. The clients require this information so that they can locate environment servers to connect to and thus they can join the VE and locate the sources of models from the brokers. Other servers can request information regarding their peers to which they can connect and mesh. Connected environment servers must register with the brokers, deregister and respond to requests for statistics that allow the brokers to maintain an up-to-date database of active environment servers. In addition, load values can be collected for the various environment servers and a limited load balancing is offered by the brokers as they direct clients to connect to servers that have a smaller load value. The brokers are also able to initiate new server creation should the load

	on the entire database become too great. When clients connect, the broker authorises the clients and provides security tokens and unique identifiers to ensure that user interaction in the environment is identifiable and secure. Lastly, statistical information and billing is possible through the brokers as user connectivity can be logged.
Server Interconnections:	The brokers must be fully meshed to ensure the consistency of their model file, user and address databases. This also ensures that no conflicting user connections occur when users connected at the same time requesting the same unique identifier.
Connection Types:	All broker servers connections are AAL5 based ABR or UBR connections with a lightweight application level acknowledgement protocol to ensure data delivery.

Table F.2: Broker Server Characteristics

Model and Media Servers (Abbreviated: MS)

Main Functions:	The MSs store the relatively large model and media files used within the VE. These files can be downloaded at any time as long as the connecting client has been authorised. The files can be pre-fetched and cached at will and the downloading of the files does not affect the services offered by the environment servers. The MSs must register with the brokers to upload a list of the available files stored on the MS, update the brokers should the list change in any way and be able to respond to requests for statistics about the MS and its usage.
Server Interconnections:	No permanent interconnections are maintained between MSs, although a MS can be requested to share data with another MS by the Brokers.
Connection Types:	All MS connections are AAL5 based ABR or UBR connections with a lightweight application level acknowledgement protocol to ensure data delivery.

Table F.3: Model and Media Server Characteristics

The Clients

Main Functions:	The clients must collect and process data from the local interaction tools and hardware, filter this data and queue it for sending to the environment servers (BS and CS). Similarly, data that arrives must be processed, the local portion of the environment database updated, and the environment rendered. If there is processing available, pre-fetching and caching of the environment data and of models that might be needed extend the functionality of the client. Further, the clients can dynamically alter their interest matrix by communicating their needs to the CS.
Connection Types:	Clients connect to the NSs, brokers and MSs using guaranteed AAL5 ABR or UBR with a lightweight application level acknowledgement protocol. They connect to the CS using a guaranteed AAL5 rt-VBR connection with an application level acknowledgment protocol and to the BS using a non-guaranteed AAL0 rt-VBR connection. The choice of the connection style is dependent on the security and reliability necessary and is discussed in more detail in the various sections relating to the servers to which the client is connecting.

Table F.4: Client Characteristics

Control Servers (Abbreviated: CS)

Main Functions:	A CS distributes portions of the environment database, in the form of descriptor files, to the clients. Once the client is interacting with the environment, the CSs mediate object control requests, creation and destruction of objects within the VE, notification of user joins or departures and the storing and updating of the descriptor files that represent the environment itself. Updates to the environment, once negotiated, are distributed via the CSs connection either with an acknowledgement guarantee, or without one, dependant on the nature of the update. The CS also calculate and distribute the interest matrices associated with specific client needs to the BS and use this matrix to distribute appropriate control information to the clients requiring this data.

Server Interconnections and Connection Types:	The CSs are fully meshed to ensure that any updates to the environment database are propagated to all CSs and thus the database consistency is ensured. These connections are AAL5 rt-VBR in nature with an acknowledgment protocol to ensure that the delivery of the data in the mesh is guaranteed. CSs connect to brokers using guaranteed AAL5 ABR or UBR connections with a lightweight application level acknowledgement protocol to ensure data delivery, and use this connection to distribute the interest matrices, and receive updates from the BSs using non-guaranteed AAL0 rt-VBR connections. Clients connect to the CSs using AAL5 rt-VBR connections that use an acknowledgement protocol to ensure data delivery.

Table F.5: Control Server Characteristics

Broadcast Servers (Abbreviated: BS)

Main Functions:	The BSs distributes update data to a selection of users within the VE. The data delivery is not guaranteed as it is temporal in nature and loss of small quantities of data is recoverable. It is RT in nature and this ensures that interaction within the VE remains immersive to the user. Data is collated before being sent in a leaky-bucket output queue and input data is filtered using interest matrices.
Server Interconnections and Connection Types:	Both servers and users are connected using AAL0 RT-VBR connections. The servers are fully meshed using individual unicast connections as update data does not need to be sent to every server, as not every broker will have interested connected clients. The BSs receive the interest matrices from the CSs using guaranteed AAL5 ABR or UBR connections with a lightweight acknowledgement protocol to ensure data delivery.

Table F.6: Broadcast Server Characteristics

The above tables list the various characteristics of the elements in the HEAVEn architecture and are a summary of the functionality, networking and interconnection characteristics for each of these elements.