January 1999

VENUS: A Virtual Environment Network Using Satellites

Sanjay Udani
University of Pennsylvania

Jord Sonneveld
University of Pennsylvania

Jonathan M. Smith
University of Pennsylvania, jms@cis.upenn.edu

David Farber
University of Pennsylvania

University of Pennsylvania Department of Computer and Information Science Technical Report No. MS-CIS-99-08.

VENUS: A Virtual Environment Network Using Satellites

Abstract

Virtual environment (VE) designs have evolved from text-based to immersive graphical systems. The next logical step of this evolution is to have a fully immersive environment in which thousands of widely distributed users will be able to move around and interact. This requires a VE architecture that can scale well for a large number of participants while providing the necessary support for quality of service, security and flexibility. Current VE architectures are unable to fully meet these requirements and a new network/protocol architecture is needed. The VENUS approach addresses these problems by creating a network architecture which is scalable and flexible. We define a new architecture consisting of a transmit-only satellite/server and bi-directional links which will be capable of sustaining a wide-area virtual environment. We then offer the preliminary results of our experiments.

Comments

University of Pennsylvania Department of Computer and Information Science Technical Report No. MS-CIS-99-08.

VENUS: A Virtual Environment Network Using Satellites*

Sanjay Udani, Jord Sonneveld, Jonathan Smith, David Farber
Distributed Systems Laboratory, University of Pennsylvania
{*udani, jsonneve, jms, farber*}*@dsl.cis.upenn.edu*

Abstract

Virtual environment (VE) designs have evolved from text-based to immersive graphical systems. The next logical step of this evolution is to have a fully immersive environment in which thousands of widely distributed users will be able to move around and interact. This requires a VE architecture that can scale well for a large number of participants while providing the necessary support for quality of service, security and flexibility. Current VE architectures are unable to fully meet these requirements and a new network/protocol architecture is needed.

The VENUS approach addresses these problems by creating a network architecture which is scalable and flexible. We define a new architecture consisting of a transmit-only satellite/server and bi-directional links which will be capable of sustaining a wide-area virtual environment. We then offer the preliminary results of our experiments.

1 Introduction

Multi-user networked virtual environments (VEs) have been studied by the graphics and networking communities for some time now. Text-based multi-user VEs have been around for many years in the form of MUDs (Multi User Dungeons), but bandwidth, processing power and architecture considerations have limited interactive, graphics oriented multi-user VEs to a small number of users.

Current graphical VE setups are lacking in several ways. First, they are not easily scalable because the amount of data traffic increases non-linearly (usually quadratically) with the number of users. Most designs only allow tens of users, with a few being able to handle up to a hundred or so users. Unless a VE system is able to consistently handle several thousand or more users interacting among themselves, creating a VE which resembles the "real" world is difficult.

Second, some of the current architectures do not provide for varying qualities of service. Every user is expected to receive the same level of service. In a large VE, this is simply not practical since there will be large differences in the capabilities of the users, from network links (modem vs. T1, for example) to processing power. Third, most of the current approaches have minimal—if any—security mechanisms. By security we mean not only the ability to know who you are interacting with but also to be able to ensure a stable virtual world that is resistant to eavesdropping and modifications by intruders. If any real interaction is to occur, users need to be able to trust and verify the information about users in the VE.

*This work is supported by the AT&T Foundation, DARPA, the Hewlett-Packard Research Grants Program, Intel Corporation, and NSF Grant #CDS-97-03220 .

The VENUS approach addresses each of these problems by creating a networked VE architecture which is scalable and flexible. The heart of the VENUS design is a server/satellite (described later in this paper) which is used to broadcast basic information about the VE to all users. VENUS scales well because the incremental cost of adding another user is minimal and fixed for the server, regardless of the amount of interaction the new user has with other members of the VE. It is flexible because it allows a wide variety of interfaces, qualities of service, and allows individual users to tailor their interactions with minimal use of the server.

The rest of this paper is organized as follows. We first take a look at some related work in the VE field. We then describe the VENUS architecture in some detail, along with its advantages. Finally we describe the experimental setup and present the initial results of the experiments. The full set of results is expected by Fall 1998.

2 Related Work

The earliest virtual environments such as IRC (Internet Relay Chat) were text based. Newer VEs can be divided by their approach to message distribution between users. Funkhouser [Funk2] provides a good summary of the various systems.

The various VE designs can be loosely organized into the four major types. The first type is based on unicast peer-to-peer messages and is used when only a unicast network is available. Whenever a user changes state, a unicast message is sent to the rest of the users. When broadcast is available, other approaches have been used. SIMNET (Simulation Network) [Calv] and DIS (Distributed Interactive Simulation) [DIS] use broadcast messages to send updates to other users participating in the VE. The broadcast approach reduces the number of messages sent out by users since each user only sends out an update once. However, each user must process the behavior of every other user to maintain consistency, and so the system does not scale beyond the capability of the least powerful participating computer. In addition, since each user broadcasts every move, the network quickly becomes saturated as the number of users increases.

NPSNET [Mace] and DIVE [Carl] are peer-to-peer systems that use multicast to send updates to a subset of users. Users/objects are mapped into multicast groups and update messages are sent only within each user's group. For example, NPSNET [Mace95c] partitions the VE into a two-dimensional grid of hexagonal cells, each of which is represented by a separate multicast group. Users send updates only within the cell in which they reside, and listen to multicasts within some radius of their cell. This approach scales well but is limited to networks which allow multicast messaging.

The systems mentioned above all have one thing in common – none of them are client-server based. Graphical client-server systems include WAVES [Kaz] and RING [Funk95]. Users (clients) do not send messages directly to other users; instead they send them to servers which forward them to other clients and servers. This method allows servers to do processing and filtering of messages before propagating them to other clients. Servers can determine if a user has made an illegal move (e.g. into a wall) or it can decide to limit sending the user's message to a subset of other users. These systems scale reasonably well, but are limited by the load that the servers can handle as every piece of user data is unicast through the servers.

3 VENUS Architecture

We envision a typical large-scale interactive virtual world to have several thousand users. A good example of a large-scale virtual world world is portrayed in the novel "Snow Crash" [Steph] which

was the inspiration for our work on VENUS. Users log into the VE and are able to attend events in large groups or interact with other users in smaller groups. We will focus on issues of networking and network access in our work and leave out the details of graphical rendering and user interfaces.

We are proposing an architecture for VEs which assumes the availability of a high bandwidth one-way satellite broadcast link and slower land-based bi-directional links between users, as shown in Figure 1. Users can be located in any geographical area that has WAN access and is reachable by the satellite transmission. The bi-directional links can be of a variety of types, ranging from modems to T1 links. In addition to the hosts and satellite, there is at least one uplink server that coordinates and organizes the overall system.

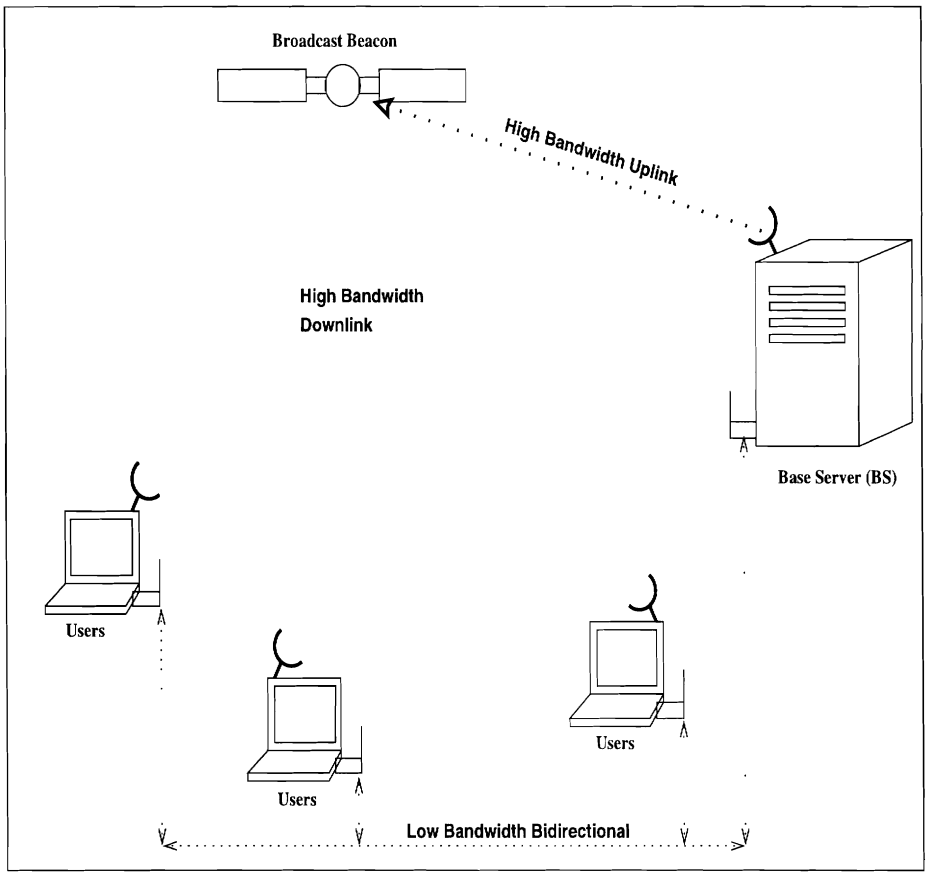

Figure 1: Satellite based VE setup.

We are proposing to use the satellite as a *location beacon* to broadcast *Object Information Packets* for the objects in the VE. The satellite will broadcast a small Object Information Packet (OIP) containing generic information for each object in the VE.

3.1 Architecture Details

Every virtual body (human, building etc.) in VENUS is an object. The owner of an object is responsible for generating an OIP (Object Information Packet) periodically and sending it to the server. This periodic update serves two functions: it informs the rest of the VE of any movement/changes in the object and also lets the server know that the object is "alive".

The server gathers the OIPs and ensures that there are no illegal OIPs before sending them to the satellite for broadcast. An illegal OIP could be from an unregistered user, or it could be for an

object that overlaps (i.e. collides with) another object. The server is also responsible for organizing the OIPs so that they are broadcast in the proper sequence based on the object's location in the VE.

3.2 The Object Information Packet (OIP)

The OIPs broadcast by the satellite provide basic information for an object. Each OIP is fixed in size and is on the order of 40 bytes. It has sufficient information for any user to gather the location and generic description of the described object, as described below.

Owner Address - the (IPv6) address of the owner of the object (16 bytes).

Object Name - allows a particular owner to have multiple objects in the VE, along with an associated name for the object given by the Object Name field ($\approx$4 bytes).

X,Y,Z-Coordinates - describe the location of the center (or some other point defined in the Generic Object Type database) of the object in the VE with reference to predefined anchor points in the VE ($\approx$12 bytes).

Orientation - defines the direction that the object is facing ($\approx$2 bytes).

Object Radius - the radius of the bounding sphere surrounding the object, used to calculate collisions and also for object scaling ($\approx$2 bytes).

Generic Object Type - gives a generic description of the object from a predefined database. The description includes simple information, such as object type and color. It is described in more detail below ($\approx$4 bytes).

Groups/Permissions - shows basic group membership and the type of user allowed access to a particular object. Described in more detail below ($\approx$1 byte).

Update - this field will inform users when the last change in the object occurred. The equivalent of marking data "stale" in a cache ($\approx$1 byte).

The Generic Object Type (GOT) field is the key field used to give a generic description of the object from a predefined database. For example, the database can have generic buildings (a cube shaped structure, for example), humans (of various builds and types) and even generic trees. This generic description gives each user the option to enjoy the entire VE without having to download detailed descriptions of each object. Users can have a complete view of the world with minimal contact with any other user, and can connect with any object owner to download details of the object they choose to. This method has advantages over most of the other schemes proposed so far, as we will see in the following sections.

The Object Radius field gives the radius of the bounding sphere (with reference to the X,Y,Z Coordinates) and serves two purposes. First, it allows the server to detect coarse-grained collisions between objects by looking at the bounding spheres of adjacent objects. Second, it allows us to scale various objects without increasing the number of GOTs. Each GOT has an associated default size which can be scaled by the Radius. For example, a default building may be a 50 foot high structure which can then be scaled to be a 200 foot building by adjusting the Object Radius field.

The Groups/Permissions field lets other users know of an object's membership in a group so that it can be granted access to some objects automatically. It can also be used to inform all users about the level of knowledge the server has of that object. For example, the server can discern between users which are highly trustworthy and those which have not been confirmed.

3.3 Frames

One OIP is needed for every object in the VE, and a full set of OIPs of all the objects in the VE is broadcast periodically by the server/satellite. This set of updates is called a *frame* and is analogous to a motion video frame.

A user does not need to listen to the entire broadcast of a frame—in the current design each frame is divided into subframes based on the VE's geography and users only need to listen to the subframes that constitute their neighborhood. This division ensures that the users are not overwhelmed by the continuous inflow of updates. Instead, they have the choice of listening to the entire VE frame only if they want to. The size of these subframes (which correspond to virtual neighborhoods) is determined by the uplink server and will vary based on the concentration of objects in any given neighborhood. A frame update header is broadcast at fixed intervals to inform users of any changes in subframe sizes.

Once users have a generic view of their neighborhood, they can then use the land-based WAN connection to get a detailed view of each object that is of interest to them. In VENUS, each user decides if a given object is of sufficient interest to actually download the detailed view. Users with a fast land-based connection may choose to get a richly detailed view of their neighborhood while users with a slow link may stay with the generic view except for the objects closest to them. A user connects with the owner of a particular object to get details of the object and they negotiate the type of data to be transferred. It may be actual video from the owner if the object owner's computer has a powerful graphics processor and the link is high bandwidth, or it may be polygon data that the user has to process locally to generate a view.

Alternatively, a DVD or CD that has the detailed graphics of the location (a standard library, so to speak) may be made available by the the owner of a site. The user then only has to obtain updates of position changes, and bandwidth usage is minimized. This option opens up many interesting possibilities—a commercial site may offer low quality graphics over the network, and sell the high quality version via DVD or CD. Users may even decide on the type of architecture they wish to see—one user may see a blue brick building whereas another user may choose to see the very same building as a pink colored metal structure.

The uplink server updates the satellite which then broadcasts the OIPs for the VE. The broadcast rate can vary based on the available bandwidth of the satellite. As the number of objects in the VE increases (or the available broadcast bandwidth decreases) this setup allows for graceful reduction of performance. For example, the frame rate can be decreased, or the server may choose to leave out non-essential objects (such as trees and bushes) based on information in its user database. The decision on what is reduced is a quality of service decision that can be made based on many different criteria.

3.4 Frame Compression

It is not necessary to broadcast every OIP in every frame—immobile objects such as buildings do not need frequent updates. We suggest a compression method where a full frame is sent at a lower frequency and moving objects are updated at a higher frequency, as shown in Figure 2. By doing this frame "compression", we are able to conserve bandwidth while maintaining system consistency. This method is similar in style to MPEG video compression.

A full frame is sent out periodically for two reasons. First, it allows users who have just joined the VE to quickly gather a view of the world, and secondly since the broadcast from the satellite is an unreliable broadcast, regular broadcasts are needed to ensure all users receive all the information.

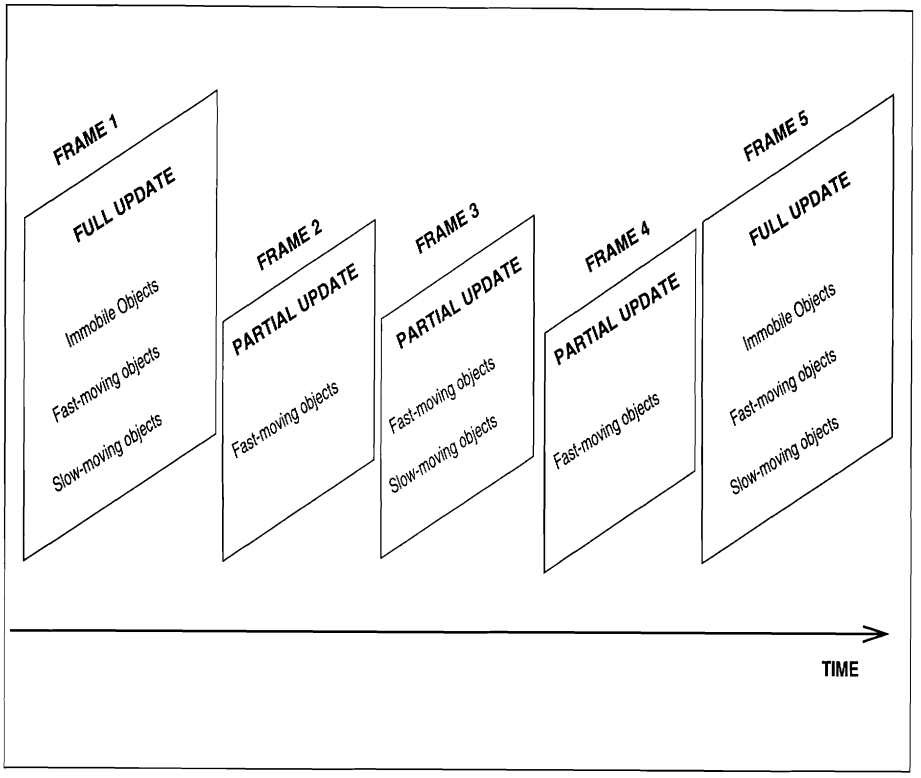

Figure 2: Compressed Frame Broadcast

4 VENUS Features

4.1 Quality of Service

In VENUS, each user is guaranteed a minimum level of service by the satellite. Regardless of the quality of the land-based connection, every user gets at least a generic view of the entire VE. If a part of a broadcast frame is corrupted or not received by users, they simply wait for the broadcast of the next frame. Since the update rate is on the order of several times a second, individual rebroadcast requests are not needed (nor possible!).

As the number of objects in the VE increases (or the available broadcast bandwidth decreases) the server can gracefully reduce performance based on predefined criteria. For example the frame rate can be decreased, resulting in slower global updates (i.e. jerkier movement). Alternatively the server may choose to leave out non-essential objects such as trees and bushes. Users can also be asked to pay a higher price to obtain a better update rate for their OIPs. These decisions can all be made and implemented by the uplink server in conjunction with the object owners.

At another level, our approach allows each user to determine their QoS individually. A user visiting a commercial site can be charged different rates for different qualities of service (e.g. better resolution, faster service, access to more information). A site owner can choose to reduce the level of service offered (e.g. loss in detailed resolution) once the number of users asking to be connected increases beyond a certain level, or the owner can simply refuse any new connection requests. The uplink server does not need to be involved in this process—it does not even need to know about the load on a particular object owner.

4.2 Accessibility

One of the problems with many of the proposed VE ideas (e.g. [Barr]) is that casual browsing is difficult. A user usually has to know the exact address of a particular object/user to find them. This is due to the limited amount of information available to all users—typically, update information is only available for objects located within a small distance of the user. This model is unfavorable since a user seeking another user cannot do so without some additional communication with a central authority of some sort (if it exists).

Also, users wanting to browse through a VE cannot do so since they have to connect with many other users before they can get a view of the VE. If the network is heavily loaded, the user will have to "travel" slowly through the area. Again, this is not a favorable situation.

In VENUS, limited information about the entire VE is broadcast and available to all users and so looking for a particular object is very straightforward—the user simply scans for matching patterns in the OIPs and contacts the owners. Browsing is also very easy since it is the user who decides what their neighborhood is and who they want to contact. A user choosing to not contact anyone can do so, and simply browse through a fully defined (albeit sparsely detailed) VE.

4.3 Security

In a typical VE, security is usually dependent on the individual users. In our approach, this load is distributed between the server and the owners of each resource. An owner has to register with the server before being allowed to send OIPs to it. Any user who sees an OIP and contacts the owner knows that the owner has to have some level of trustworthiness (or at least accountability) before being allowed into the VE.

Users wanting a higher level of verification than provided by the server can negotiate with each owner that they contact. Since higher levels of detail are only provided on a per-connection basis, the user (or owner) can control the kind of access offered to each user.

4.4 Scalability

In evaluating scalability, let us first consider current satellite technology. For example, DirecPC, a satellite-based network access system is currently available and offers downlink rates of up to 10 Mb/s. Other systems such as SPACEWAY by Hughes Communications are in the planning stages and will offer data rates of 108 Mb/s or more [Seidm] when deployed. A 108 Mb/s broadcast can potentially accommodate a VE with 10,000 objects at over 30 uncompressed frames per second. As satellites increase their available bandwidth in the future, VENUS-based VEs will be able to handle an even larger number of objects since the incremental bandwidth required per object is a fixed number (the size of the OIP). There is a linear correlation between the increase in broadcast bandwidth usage and the number of users/objects.

The second potential bottleneck in the system is the server network connectivity and processing capability. As the number of objects (and object owners) increases, the number of OIP updates that the server has to handle will increase proportionally. Bandwidth and processing capability will at some point become major bottlenecks. A possible solution would be to have multiple layers of servers, each filtering out unnecessary traffic and sending a stream of broadcast-ready OIPs to the main server. The tradeoff of course is in added latency.

In our experiments we will explore the performance that can be expected from varying broadcast bandwidths. We will use analysis and simulations to show expected performance levels for typical VE setups. We will also show the expected load capability of a server for various levels of processor power and network connectivity.

4.5 Latency

Typical satellite latencies can vary greatly, from 400 ms for high altitude satellites to under 50 ms for LEOS' (Low Earth Orbit Satellites). In some computer use (email, ftp, browsing), latency is not a critical issue but in an immersive interactive environment it can cause major problems, especially when the users of the VE are distributed across a wide geographical area. One advantage of a satellite broadcast is that all users do receive the broadcast essentially at the same time, and this can be used to synchronize events globally.

The speed of light puts a limit on reducing latency, but there are some approaches which may reduce the effects of latency in an interactive environment. For example, motion estimation (*dead reckoning*) is used by some VEs [Mace95a] to reduce bandwidth usage but it could also potentially be used to limit the effects of latency if the server could predict collisions and warn users before they actually collided.

5 Experimental Setup

To prove the key features of the VENUS architecture, we have set up a network of computers as shown in Figure 3. The machines we are using are 3 200MHz Dual Pentium Pros (as clients) and one 300 MHz Pentium II (server). All computers are equipped with 384 MB or more of DRAM and dual 3Com 3c905b 100 Mbps Ethernet cards. The machines are connected via two 3Com Hubs forming two distinct subnets. All are running RedHat Linux 5.1 with an unmodified 2.1.106 SMP kernel.

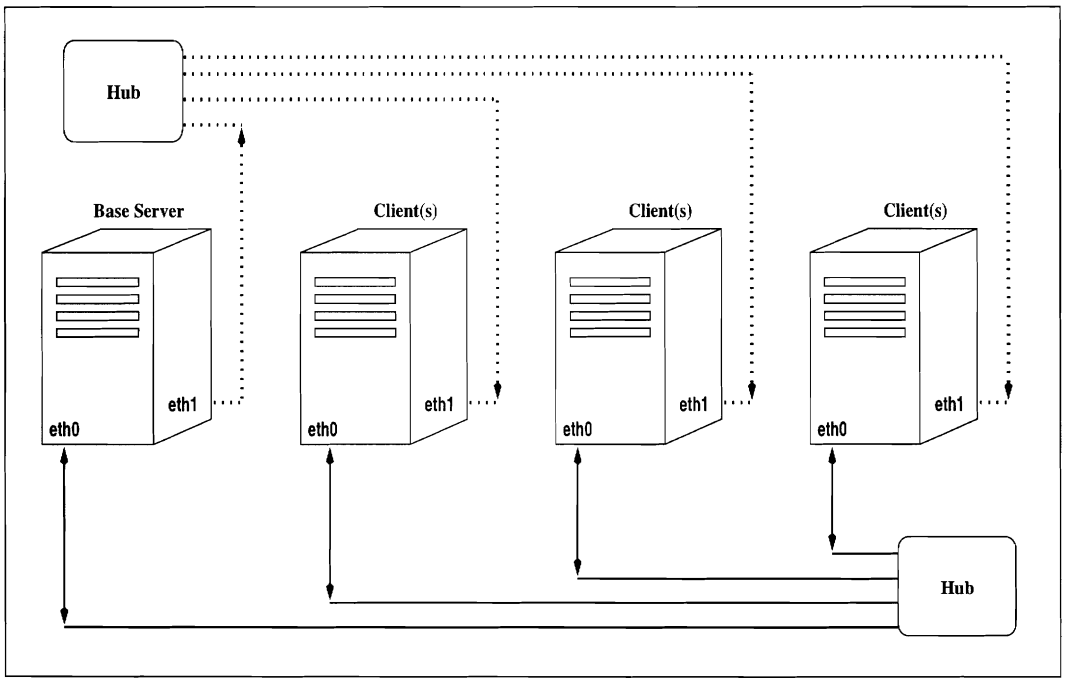

Figure 3: Experimental Setup

One subnet (eth1) is used only by the server and emulates the satellite broadcast. The other subnet (eth0) is used by the client machines for sending OIPs to the server. The server acts as both the server and broadcast satellite for the system since in the architecture the server and satellite are one logical entity. The OIPs are sent using UDP.

The server maintains the VE world, which is an array of all the OIPs in the system. In the current setup the server takes the OIPs it receives from the clients and stores them in the array indexed by their client/OIP id, but later versions will have hash tables to store the OIPs by their location in the VE. The clients have their own arrays to maintain the objects they own.

Our objective is to show that the setup can scale (i.e. handle an increase in users/objects). Based on the server load and bandwidth usage, we will then analyze it to see if there are any limiting factors in extrapolating the results to a geographically wider setup.

6 Preliminary Results

To see how many objects the server could handle, we set up each of the client machines to send $\frac{1}{3}$ of the total number of OIPs in our VE world. In the case of a 30,081-object world, each client generated OIPs for 10,027 objects. In a real VENUS setup each client would typically own only tens of objects. This overload of the client in our experiment does not affect the results as we are measuring the server load.

Figure 4 shows the results of the preliminary experiments. With no OIP compression (i.e. all OIPs broadcast at the same rate) we obtained a broadcast frame rate of 10.6 frames/sec for a world of 30 081[1] objects. The uncompressed frame rate is shown by the solid line in both graphs.

To show the effects of compression, we divided the broadcast world into 4 equal parts. One part was broadcast at the normal rate and the other three were broadcast at ratios of $\frac{3}{4}X, \frac{1}{2}X, \frac{1}{4}X$ of the first part. In other words, the fourth part was broadcast once for every four broadcasts of the first part. This follows from the discussion in Section 3.4 where we categorize objects in several groups ranging from stationary objects that need infrequent updates to fast-moving objects that need high update rates. The results are shown by the dotted lines in Figure 4 indicate that this compression technique works quite well, especially with average world sizes. In the case of the 30,081 object world, the frame rate rises from 10.6 frames/sec to 19 frames/sec. These preliminary results indicate that a fairly large VE can be sustained at high frame rates.

Once we do more processing on the server – such as hashing by location and collision detection – we expect the load to increase somewhat but this can be offset by using a more powerful system for the server.

Transmit rate (frames/sec)		10	20	50	max.
Total Client Load with	37 OIPs	33%	33%	34%	70%
	370 OIPs	35%	35%	36%	70%
	1480 OIPs	39%	44%	45%	71%

Table 1: Client load as a function of frame rate and OIP count.

We also measured the load on the processor of a client that was listening to the full 100 Mbps broadcast, filtering out relevant OIPs and sending back updates for objects that it owned. For this experiment, we had two of the client machines sending most of the OIPs, with the client we were measuring filling in the rest. For example, when the client we were measuring was handling 37 objects of the 30,007 object world, the other two clients were set up to handle $\frac{30007-37}{2}$ objects each.

[1]The seemingly odd values for world sizes are a result of the MTU of the ethernet link for the experiment, and not from the architecture.

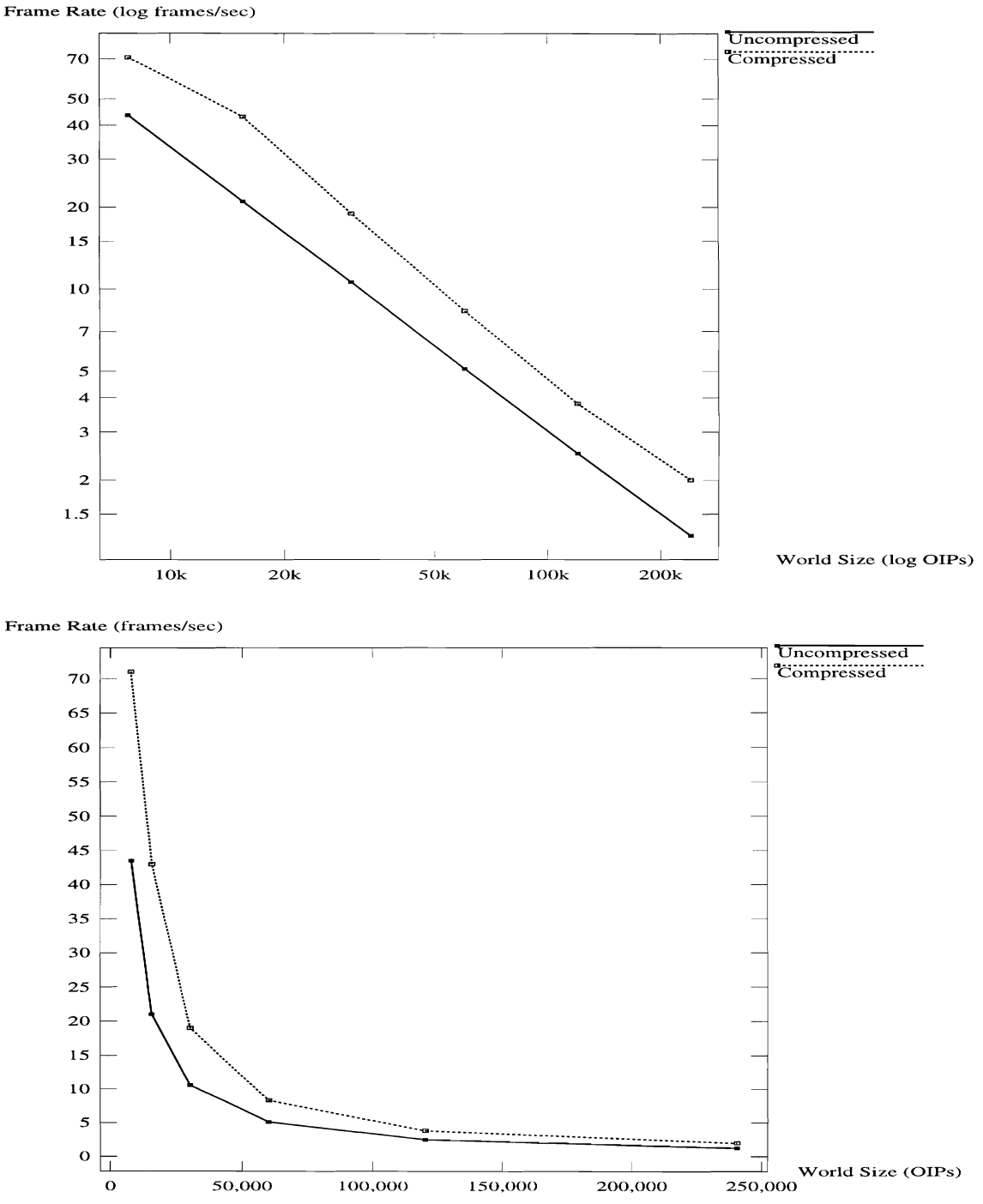

Figure 4: Frame Rate vs. World Size

Table 1 shows the load put on the client when varying transmit rates as well as varying the number of OIPs owned by the client. The clients were not loaded with other tasks (beyond the usual system tasks) during the experiments.

Table 1 shows that for any reasonable transmit rate (i.e. updating objects at up to 50 times per second) and average object ownership (tens of objects per client), just one third of the total CPU time is used in filtering the broadcast and sending out updates. Only when we set the transmit rate to maximum (essentially flooding the network) does the CPU load go above the $\frac{1}{3}$ threshold. Although we expect the typical client to own only tens of objects (and so generate only tens of OIPs), we ran our experiments with higher OIP counts to see the loading effects.

6.1 Research Issues and Contributions

Some of our ideas that we believe are novel in their application to this problem include:

- The use of a beacon to maintain and broadcast the state of the world to all the users in the VE along with the separation of attribute information and individual interaction details by the use of Object Information Packets (OIPs) to give a generic, low-bandwidth view of the world to all users. This in turn allows for high scalability because the incremental cost of adding a user is minimized.
- Each object (and owner) decides what load level it wishes to handle, without affecting the global view of that object or any intervention by the server. This allows for varying qualities of service on a per-connection basis.
- Each user can have a *complete* view of the world with minimal contact with any other user. In fact, a user could simply listen in to the broadcast and see the whole VE world.

7 Summary

We have described a new architecture for a scalable, wide-area, interactive virtual environment which offers several advantages over current systems. VENUS is a hybrid of peer-to-peer, multicast, broadcast and client-server VE designs. It broadcasts attributes (location, size etc.) of users and uses peer-to-peer or multicast methods for the exchange of user data (voice, video, graphical details etc.). A server is used in keeping track of the overall world.

VENUS is intended to scale not only for a large number of users, but also for users that are geographically distributed over a large area. Our design takes advantage of currently available technologies such as satellites and it is possible to deploy the VENUS architecture now. VENUS spreads the cost of a satellite channel across all the users thus minimizing cost while maximizing efficiency. Our preliminary experimental results support our thesis.

References

[Barr] J. W. Barrus, et. al., *Locales and Beacons: Efficient and Precise Support for Large Multi-User Virtual Environments*, Mitsubishi Electric Research Laboratory Technical Report TR95-16a. August 1996.

[Calv] J. Calvin, A. Dickens, et. al., *The SIMNET Virtual World Architecture*, Proceedings of the IEEE Virtual Reality Annual International Symposium, September 1993, pp. 450-455.

[Carl] C. Carlsson and O. Hafsand, *DIVE: A Multi-User Virtual Reality System*, Proceedings of the IEEE Virtual Reality Annual International Symposium, September 1993, pp. 450-455.

[Funk95] T. Funkhouser, *RING: A Client-Server System for Multi-User Virtual Environments*, ACM CSIGGRAPH Special Issue on Symposium on Interactive Graphics, Monterey CA, 1995, pp. 85-93.

[Funk2] T. Funkhouser, *Network Topologies for Scalable Multi-User Virtual Environments*, AT&T Bell Labs.

[DIS] IEEE International Standard, ANSI/IEEE Std 1278-1993, *Standard for Information Technology, Protocols for Distributed Interactive Simulation*, March 1993.

[Kaz] R. Kazman *Making WAVES: On the Design of Architectures for Low-end Distributed Virtual Environments*, Proceedings of the IEEE Virtual Reality Annual International Symposium, September 1993, pp. 443-449.

[Mace95a] M. Macedonia, et. al, *Exploiting Reality with Multicast Groups: A Network Architecture for Large-Scale Virtual Environments*, IEEE Computer Graphics and Applications, September 1995, pp. 38-45.

[Mace] M. Macedonia, M. Zyda et. al., *NPSNET: A Network Software Architecture for Large Scale Virtual Environments*, Presence, Vol. 3, No. 4.

[Mace95c] M. Macedonia, M. Zyda et. al., *A Network Architecture for Large Scale Virtual Environments*, Proceedings of the 1995 IEEE Virtual Reality Annual Symposium, March 11-15 1995, RTP, North Carolina

[Seidm] L. P. Seidman, *Satellites for Wideband Access*, IEEE Communications Magazine, October 1996, pp. 108-115.

[Steph] N. Stephenson, *Snow Crash*, Bantam Books, 1993.